VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Technologie PhysX

PhysX Technology

2011

Vítězslav Tkáč

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2011 .....

Chtěl bych poděkovat vedoucímu mé diplomové práce Ing. Janu Platošovi, Ph.D. za pomoc a čas, který mi věnoval. Velký projev díků patří i mým rodičům, kteří mě po celou dobu studia psychicky i finančně podporovali.

Abstrakt

Tato diplomová práce se zabývá popisem architektury PhysX enginu. Vývojem jednoduchých grafických aplikací pomocí této technologie a testováním výkonu na různých procesorech a specializované grafické kartě.

Klíčová slova: PhysX, PhysX SDK, fyzika, fyzikální engine

Abstract

This diploma thesis deals with the description of the PhysX engine architecture. The development of simple graphical applications using this technology and performance testing on different processors and dedicated graphics card.

Keywords: PhysX, PhysX SDK, physics, physics engine

Seznam použitých zkratek a symbolů

ASCII	–	American Standard Code for Information Interchange
CPU	–	Central Processing Unit
CUDA	–	Compute Unified Device Architecture
DME	–	Data Movement Engine
FPE	–	Floating Point Engine
FPS	–	Frames Per Second
GLUT	–	OpenGL Utility Toolkit
GPU	–	Graphic Processing Unit
OpenGL	–	Open Graphics Library
PCE	–	PPU Control Engine
PPU	–	Physics Processor Unit
RISC	–	Reduced Instruction Set Computer

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Fyzika. Jen velmi málo lidí na planetě bude tvrdit, že se s tímto slovem řeckého původu nikdy nesetkalo. S fyzikou se setkáváme dennodenně, aniž bychom si to občas uvědomovali. Každá naše činnost, přírodní jev i klidový stav podléhá fyzikálním zákonům. Její důležitost si uvědomovali už ve starověkém Řecku. Tehdy bylo zdrojem poznání úvaha a vypozorované zkušenosti. Přesto byla Aristotelova fyzika vrcholem poznání po tisíc let. Tyto prvopočáteční úvahy daly prvotní impuls vývoji vědě, která se významnou měrou podílí na vývoji lidské civilizace.

V dnešní době informačních technologií se stále častěji prolínají nabyté znalosti fyziky a jejich zákony s poměrně mladým oborem informatiky - počítačová grafika. Naše monitory, displeje a jiná zobrazovací zařízení zaplavují spousty grafických animací různých úrovní od jednoduchých po velmi propracované napodobující dění reálného světa. Dnešní uživatelé aplikací založených na grafice kladou stále větší nároky na dojem skutečné reality, což nutí vývojáře do aplikací zakomponovat více detailů a fyzikálních výpočtů.

Pro uspokojení uživatelů se na trhu objevují specializované softwarové nástroje pro výpočet fyzikálních jevů těles, kapalin nebo plynů. Jejich distribuci umožnil mimo jiné vývoj hardwarového vybavení osobních počítačů, jelikož jsou aplikace tohoto typu náročné na výpočetní sílu počítače.

Úkolem mé diplomové práce je prozkoumat jeden z těchto nástrojů zvaný PhysX, seznámit se s jeho historií a vnitřní strukturou. Dalším úkolem je nabytí praktických znalostí při implementaci několika jednoduchých příkladů. Na závěr pomocí vytvořených aplikací porovnat běh aplikace na CPU a grafické kartě podporující PhysX.

2 Souhrn oblasti fyzikální akcelerace

V současnosti je vyvíjena celá řada nástrojů pro výpočet simulace fyzikálních jevů s různými klady a zápory a různým stupněm propracovanosti. Fyzikální enginy lze rozdělit na dvě hlavní skupiny podle způsobu použití:

- Vysoce přesné – tento druh fyzikálního jádra se vyznačuje vysokou přesností fyzikálních simulací. V protikladu jsou náročnější na výpočetní výkon a čas. Používají se pro vědecké účely a vytváření počítačových animací [1]. Mezi zástupce této skupiny lze zařadit:
  - VisSim – je vizualizační prostředí integrované s výpočetním programem Mathcad. Používá se nejen v průmyslu pro řešení leteckých, elektrických, hydraulických, mechanických a dalších systémů, ale také pro vědecké účely.
  - Working Model – simulační software mechanických komponent jako jsou lana, pružiny a motory s různými tělesy ve 2D prostoru.
- Výpočty v reálném čase – nejsou tak přesné jako enginy předchozí kategorie, ale jsou přizpůsobeny pro simulaci fyzikálních jevů v reálném čase. Jejich využití spočívá především v počítačových hrách [1]. Vzhledem k velké oblibě her má tento model širší škálu zástupců, což vede k většímu konkurenčnímu boji. To má za následek rychlejší vývoj této třídy fyzikálních simulátorů a zvyšování jejich kvalit.
  - PhysX je multiplatformní fyzikální jádro napsané v jazyce C++, optimalizované pro hardwarovou akceleraci PPU karet a grafických karet firmy nVidia. Ta je současným vlastníkem této technologie.
  - Havok byl v počátcích vyvíjen stejnojmennou irskou firmou. Primárně byl určen pro grafické adaptéry firmy ATI, ale od roku 2008 je vlastníkem firma Intel. Snad i proto je Havok optimalizován pro běh na procesorech. V dnešní době je jedním z největších konkurentů PhysX enginu. Na rozdíl od PhysX má placenou licenci což ho znevýhodňuje v konkurenčním boji.
  - Bullet je fyzikální engine s open source licencí simulující kolize pevných a měkkých těles. Mimo použití v počítačových hrách nachází uplatnění při vytváření vizuálních efektů ve filmech.
  - Box2D je open source simulátor fyziky ve 2D prostoru. Implementován je v jazyce C++ a je využíván především pro hry založené na technologii Flash. Objevuje se i na platformách iPhone nebo Android.
  - Physics2D.Net – jak již napovídá název je spojen s prostředím .NET 2.0 a vyšší. Implementován je v jazyce C# a optimalizován pro velké množství objektů.
  - Phyz – fyzikální engine s vestavěným editorem DirectX grafiky a zvuku.
  - Dymix – realizuje simulaci pevných těles na mobilní technologii J2ME.

3 Historie technologie PhysX

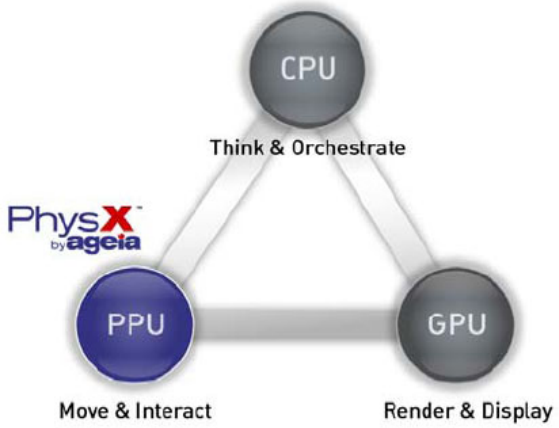
3.1 Vznik a vývoj PhysX technologie

Původně byla PhysX technologie známá jako fyzikální engine NovodeX, který byl vyvíjen švýcarskou firmou Novodex AG od roku 2002. Od srpna tohoto roku byla zveřejněna první dostupná verze s označením 1.4.

V roce 2004 získala firmu Novodex AG (personál i celý softwarový vývoj) americká firma Ageia, která se mimo jiné zabývala vývojem dosud příliš nepoužívané periférie PPU. V době odkoupení byl engine NovodeX SDK ve verzi 2.3 a byl přejmenován jako fyzikální engine PhysX SDK. V této verzi je dostupný mimo platformy Windows XP také konzole Microsoft XBox360 a Sony Playstation 3 [2].

Rok poté uzavřela Ageia akvizici se švédskou vývojářskou společností Meqon research AB o převzetí jejich multiplatformní technologii Meqon, jež rovněž zakomponovala do svého vyvíjeného enginu. Spojení těchto technologií slibovalo poměrně velké úspěchy a pokrok ve fyzikální grafice, a proto byla licence na PhysX SDK placená [3].

Pro potřeby tohoto nového enginu vyvinula Ageia PPU kartu, která se specializovala na výpočty fyzikálních jevů (kolize objektů, simulace kapalin). Filosofie této periferie spočívala v doplnění v té době ne příliš specializovaných procesorů (CPU) o tyto hardwarové akcelerátory a tím navýšit výkon podporovaných aplikací (obrázek 1) [4]. Tento čip však neměl příliš velký úspěch. Příčiny nezájmu o tuto technologii byl kromě vysoké vysoké ceny fakt, že herní svět, respektive většina dostupných her nebyly na takové úrovni, aby tuto technologii využily. Na základě tohoto neúspěchu změnila vlastnící firma obchodní strategii a uvolnila PhysX SDK jako volně šiřitelné pro vývojáře her a aplikací.



Obrázek 1: Architektura dle Ageii [4]

Ani tento krok výrazně nepomohl zlepšit situaci nezájmu, a proto v roce 2008 koupil tuto firmu gigant mezi dodavateli grafických čipů nVidia. Ta PhysX engine převzala a zakomponovala jej do ovladačů svých grafických karet nové generace a od verze 2.8.1

pokračuje s vývojem nástrojové sady s podporou původního akcelerátoru. Fakt, že již nebylo nadále potřeba dalšího drahého hardware s velmi omezeným využitím a vývojové nástroje byly po registraci volně dostupné, byly odstraněny chyby, kterých se Ageia ve své firemní strategii dopustila. Ani dnes nemůžeme hovořit o masovém používání této technologie, přesto si PhysX zajistila své místo na trhu a v porovnání s konkurencí můžeme hovořit o špičce mezi herními fyzikálními enginy [5].

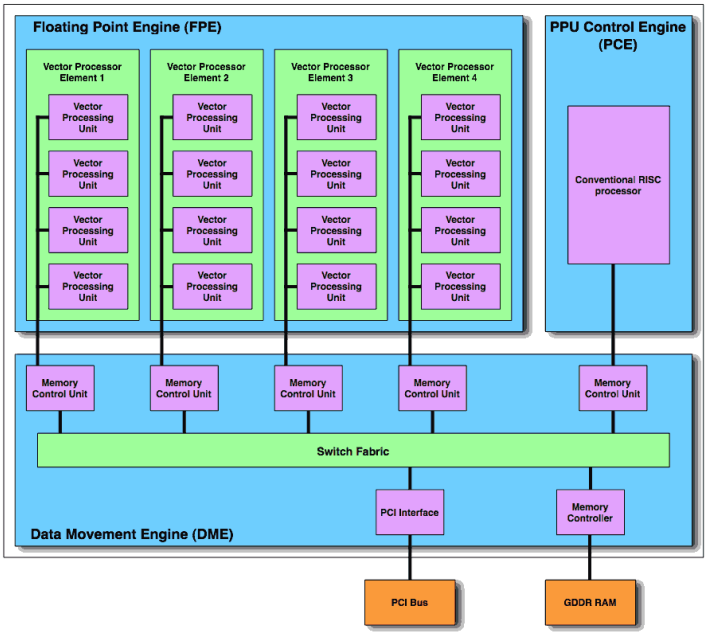
3.2 Vznik a vývoj PPU

PPU je periférie se speciálním procesorem navrženým pro výpočty fyzikálních algoritmů, dnes již stále častěji používané v moderní grafice nejen herního průmyslu. Nutnost fyzikálních výpočtů vzniká například při kolizích pevných či měkkých objektů, simulací kapalin, tkanin, lámání objektů nebo metoda konečných prvků používané pro vědecké a průmyslové účely.

Prvními PPU s označením SPARTA a HELLAS vznikly jako univerzitní projekty, které zatím nejsou masivně používány veřejností.

O hromadnou distribuci se zasloužila firma Ageia, jejichž první periférii této kategorie s označením PhysX P1 výrobila firma ASUS (na základě vývoje a dodávky čipu firmy Ageia) a v roce 2006 byla uvedena na trh. Tato karta podporovala pouze fyzikální engine PhysX, tudíž měla velmi omezenou použitelnost. Hardware byl navržen pro standardní sběrnici PCI, přestože se na trh začal prosazovat nový typ sběrnice PCI Express a 130Nm technologie spolu s 125 miliony tranzistorů slibovalo 20 miliard instrukcí za vteřinu [6].

Blokové schéma je uvedeno na obrázku 2.



Obrázek 2: Schéma PhysX P1 [7]

Karta je rozdělena na tři základní bloky:

- PCE — v této části je umístěn hlavní řídící blok periferie
- FPE — část pro výpočty s plovoucí desetinnou čárkou
- DME — blok zabývající se přenášením dat mezi pamětmi

PCE blok je vybaven procesorem s redukovanou instrukční sadou. Jeho hlavní funkcí je řízení zbylých dvou bloků a komunikace s vnějším systémem. Na společné sběrnici (DME) je umístěno mimo interní 128MB paměti také pět MCU. Čtyři z nich uskutečňují přenosy dat z FPE bloku, kde jsou umístěny jednotlivé výpočetní jednotky. Každá z nich disponuje malou pamětí což se sdílenou pamětí tvoří dvojitou vyrovnávací paměť. Pátá MCU jednotka obsluhuje PCE blok [7].

4 Popis architektury PhysX

PhysX je software schopný simulovat jednoduché i složitější fyzikální systémy jako je fyzika pevných těles, kapalin, tkanin, tvorba silových polí, simulace vozidel a další. V této kapitole se budu zabývat strukturou systému PhysX jako takovou. Software je napsán v programu C++.Jedná se o ucelený balík tříd rozdělených do pěti základních komponent. Seznam všech tříd a jejich provázání je značně rozsáhlé, tudíž není předmětem této práce zde uvést všechny. Zmíním se tedy o stěžejních třídách s jejich zjednodušenými třídními diagramy za účelem pochopení principu základní implementace. Literární prameny pro tuto kapitolu pochází z reference [8].

4.1 Komponenty

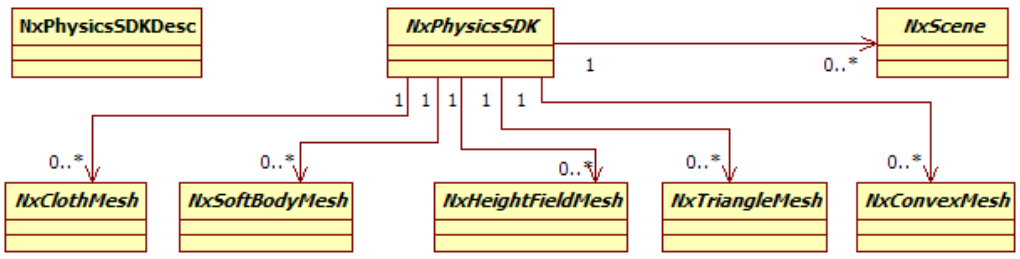
Jednotlivé komponenty jsou balíky tříd rozdělené podle své funkčnosti, která je pro ně typická. Mezi hlavní komponenty patří:

- Physics SDK (fyzikální komponenta) implementuje chování pevných, pružných, tekoucích těles.
- Cooking SDK se používá pří vytváření nových nedefinovaných objektů a vytváří pro ně systém detekce kolizí.
- Foundation SDK obsahuje matematické a podpůrné třídy.
- Charakter SDK.
- PhysXLoader – pomocí této komponenty se načítá jádro fyzikální komponenty.

4.2 Třídy Physics SDK

4.2.1 NxPhysicsSDK

Základním stavebním prvkem PhysX aplikací je třída NxPhysicsSDK. Jedná se o abstraktní třídu návrhového vzoru Singleton. To znamená, že v celém programu běží jediná instance této třídy.



Obrázek 3: Třídní diagram *NxPhysicsSDK*

Třída obsahuje funkce pro vytváření instancí dalších tříd a uchovává si jejich ukazatele. Nejdůležitější vytvářené objekty jsou:

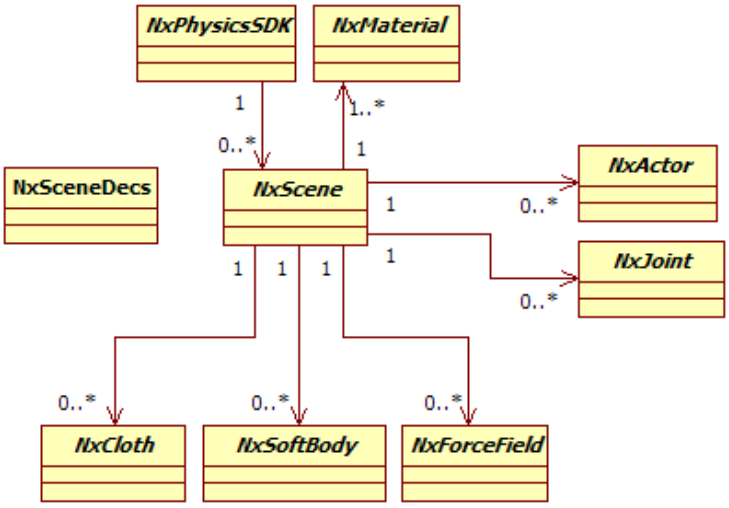
- *NxScene* — tzv. scéna na níž se odehrává veškeré dění dle diagramu je patrné, že pro jednu instanci *NxPhysicsSDK* lze vytvořit více scén, ale ty se nemohou mezi sebou nijak ovlivňovat.
- *NxTriangleMesh* – objekt s daty trojúhelníkové sítě, které lze opakovaně použít pro vytváření rozsáhlých objektů.
- *NxClothMesh* – opět data trojúhelníkové sítě, sloužící pro vytváření tkanin.
- *NxSoftBodyMesh* – objekty znovupoužitelných dat pro vytváření měkkých těles.

Dále slouží k nastavení globálních parametrů, které ovlivňují všechny scény vytvořené v rámci této instance.

4.2.2 NxScene – scéna

Na scéně se odehrává veškeré dění. Jako analogie z reálného světa i zde vystupují aktéři různých typů, tvarů, materiálu. Scéna simuluje vzájemné interakce mezi aktéry, jejich chování v prostoru a čase. Kromě aktérů se na scéně mohou vyskytovat různé typy světelných zdrojů, které navozují celkovou atmosféru. Takováto scéna by neměla žádný smysl pokud by ji nesnímala kamera. Instance třídy NxScene je složitý objekt mnoha funkcí např.:

- Vytváření a odstranění instancí aktérů.
- Filtraci kolizí a vytváření kolizních skupin.
- Vytváření a správa materiálů.

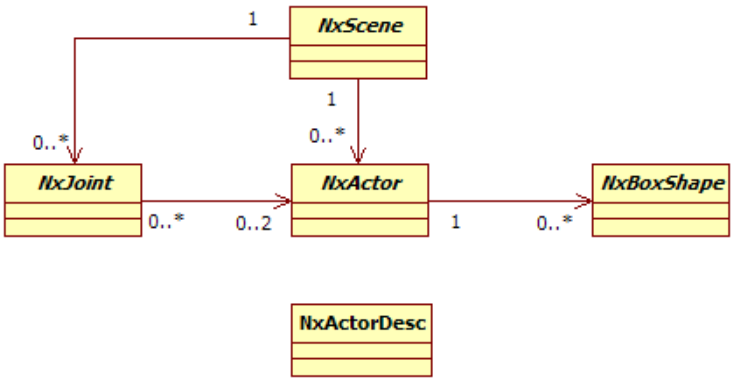


Obrázek 4: Třídní diagram *NxScene*

4.2.3 NxActor – aktéři

Aktéři jsou hlavními představiteli 3D simulace. Jedná se o objekty umístěné na scéně ať skryté nebo viditelné. Každý z nich je definován svými vlastnostmi a chováním.

Aktér je vytvářen v rámci jediné scény a nemůže se objevit ve více scénách najednou. Při vytváření se používá popisná třída typu *NxActorDesc*, která obsahuje veškeré parametry potřebné pro jeho vytvoření. Tělo aktéra je tvořeno jedním nebo několika tvary. Ty jsou definovány pomocí třídy *NxShape* a jejich potomků.



Obrázek 5: Třídní diagram *NxActor*

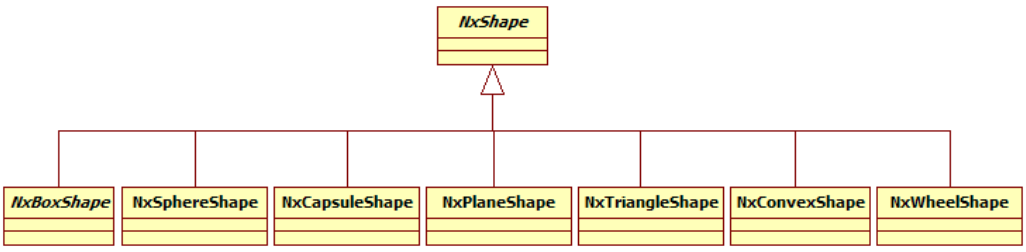
V třídním diagramu je pro zjednodušení zobrazena pouze vazba mezi třídou *NxActor* a *NxBoxShape*. Tato vazba platí pro všechny potomky rodičovské třídy *NxShape* (viz obrázek 6).

4.2.4 NxShape – tvary

Tvary se používají pro definování vzhledu, simulaci kolizí a do jisté míry chování aktéra. Instance této třídy se použije jako jeden z parametrů k vytvoření aktéra. Pro potřeby zahrnout co nejvíce geometrických útvarů jsou definovány následující typy.

- krychle, kvádr (*NxBoxShape*),
- koule (*NxSphereShape*),
- kapsle (*NxCapsuleShape*),
- síť trojúhelníků (*NxTriangleMeshShape*),
- komplexní obal (*NxConvexMeshShape*),
- výškové pole (*NxHeightField*),
- plocha (*NxPlaneShape*),
- kolo (*NxWhellShape*).

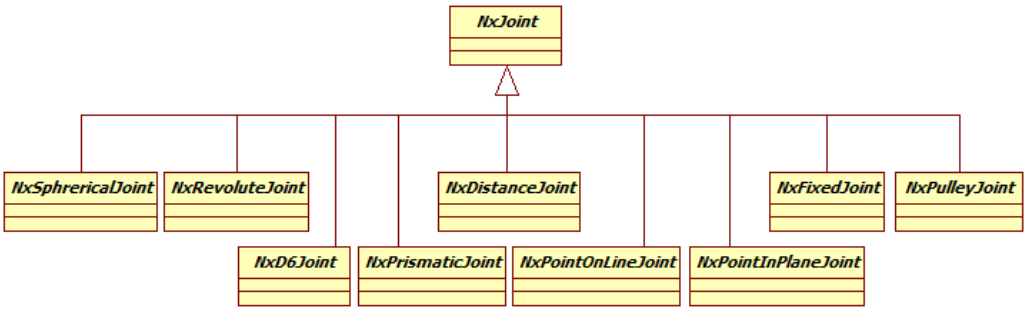
Všechny tvary mají společného předka *NxShape*.



Obrázek 6: Třídní diagram *NxShape*

4.2.5 NxJoint – spoje

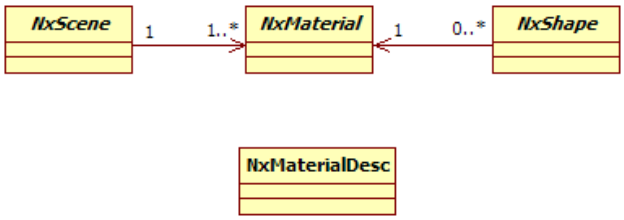
Spoje se používají ke spojení dvou aktérů v jeden celek. Nejedná se však o typ spojení kdy v rámci jednoho aktéra spojujeme více útvarů do jednoho. Pro tento spoj (podobně jako u tvarů) se vytváří instance třídy, která dědí z obecné třídy *NxJoint*.



Obrázek 7: Třídní diagram *NxJoint*

4.2.6 NxMaterial – materiály

Materiály jsou nedílnou součástí objektů vyskytujících se na scéně. Ovlivňují jejich vlastnosti a přibližují jejich chování realitě, pro kterou platí fyzikální zákony za všech okolností. Do technologie PhysX je systém materiálů implementován dle následujícího diagramu.



Obrázek 8: Třídní diagram *NxMaterial*

Instance materiálů jsou vytvářeny a indexovány z objektu scény. Při inicializaci se udává jako parametr popisná třída *NxMaterialDesc* pomocí které se předávají veškeré parametry.

4.3 Třídy Foundation SDK

4.3.1 NxMath

NxMath je statická třída obsahující balík funkcí pro skalární matematické výpočty s plovoucí desetinnou čárkou (nalezení minima, maxima, výpočet logaritmů, goniometrické funkce a další). Dále definuje významné matematické kostanty a datové typy.

4.3.2 Datové typy a struktury

Definice 4.1 *Datové typy jsou druhy proměnných a konstant, které jsou určeny svým oborem hodnot a operacemi, které lze s jejich daty provádět [9].*

Vývojový nástroj PhysX sjednocuje číselné datové typy jazyka C++ podporovaných platforem pro zlepšení přenositelnosti aplikací mezi jednotlivými platformami.

datový typ	původní datový typ (MS Windows)	původní datový typ (Linux)
NxI64	__int64	long long
NxI32	signed int	signed int
NxI16	signed short	signed short
NxI8	signed char	signed char
NxU64	unsigned __int64	unsigned long long
NxU32	unsigned int	unsigned int
NxU16	unsigned short	unsigned short
NxU8	unsigned char	unsigned char
NxF64	double	double
NxF32	float	float

Tabulka 1: Číselné datové typy

„Abstraktní datová struktura je způsob, jak efektivně uložit data tak, aby práce s nimi byla relativně snadná. Je to abstraktní skladiště pro data definovaná v rámci množiny operací a pro výpočetní složitosti při vykonávání těchto operací, bez ohledu na implementaci v konkrétní datové struktuře [10].“

Rámec PhysX definuje hned několik datových struktur mezi nejdůležitější patří:

- NxArray jednoduchý zásobník pro evidenci objektů stejné třídy.
- NxVec3 je datová struktura vektoru složená ze tří číselných prvků, tedy matice 1×3.
- NxMat33 uchovává matici 3×3.
- NxMat34 struktura kompletní afinní transformace ve 3D prostoru složená z matice 3×3 a vektoru 1×3.
- NxQuat tzv. quaternion, uspořádaná čtveřice složená ze čtyř číselných prvků nebo jednoho skaláru a jednoho vektoru.

5 Tutoriál PhysX technologie

5.1 Podpora systémů a vývojové nástroje

V současné nejnovější verzi PhysX 2.8.4 lze vyvíjet aplikace v platformách:

- MS Windows XP a výše — vývojový nástroj pro tuto platformu doporučuje výrobce software z řady MS Visual Studio. Tento software jsem používal při implementaci své práce konkrétně MS Visual Studio 2008 v.9.0.
- MS XBox360 — pro tuto platformu je doporučován vývojový nástroj Xbox Development Kit (XDK) od verze v6534
- OS Linux — pro překlad je vyžadována knihovna gcc ve verzi 3.3 a vyšší.
- Sony Playstation 3.

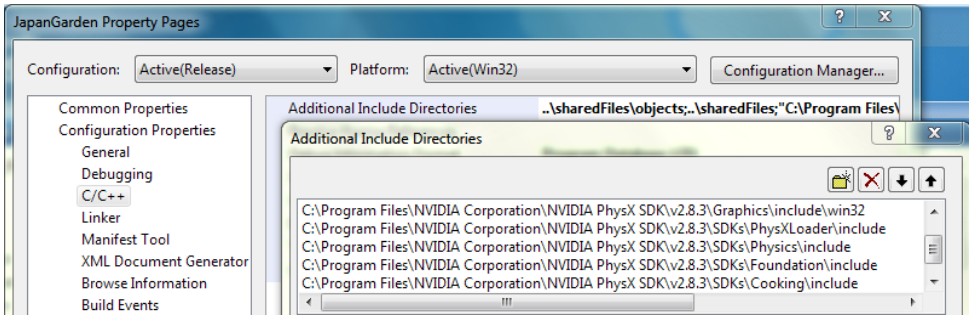
Jelikož jsem ve své práci pracoval pouze s operačním systémem MS Windows 7 bude se následující text zabývat pouze implementaci v tomto systému. Při tvorbě kapitoly 5 jsem použil vědomosti nabyté z pramenu [8].

5.2 Instalace a import

Pro tvorbu vlastních aplikací je nutné nainstalovat následující nástroje:

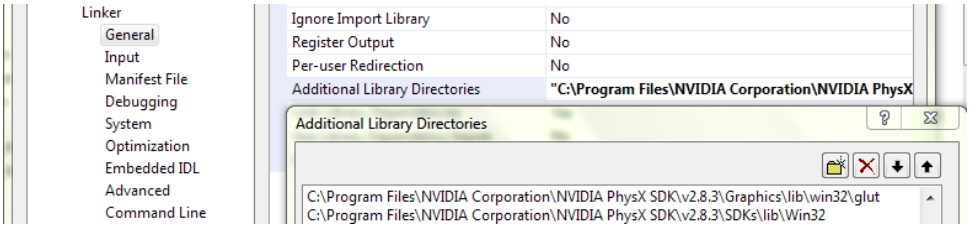
- PhysX system software – podpora pro běh programů s PhysX.
- PhysX SDK – obsahuje hlavičkové soubory, knihovny v jazyce C++.

Jak již bylo zmíněno k implementaci jsem používal Visual Studio 2008 jazyk visual C++. Po vytvoření projektu (zcela prázdného, bez předkompilovaných hlavičkových souborů) jsem importoval následující knihovny uvedené na obrázku 9.



Obrázek 9: Importované knihovny

Význam jednotlivých komponent je zmíněn o kapitolu výše. Tímto bylo vyřešen import hlavičkových souborů. Kromě nich je pro úspěšný překlad aplikace vyžadovány statické knihovny. Ty jsem pro sestavovací program (Linker) připojil dle obrázku 10.



Obrázek 10: Knihovny pro sestavovací program

Kde do položky Linker – Input jsem vložil jména jednotlivých knihoven tedy:

- PhysXLoader.lib – tato knihovna načítá jednotlivé moduly.
- glut32.lib – knihovna grafického rozhraní OpenGL.

Literární pramen importu pochází z reference [11].

5.3 První program

Prvním programem jsem pouze ověřil správné připojení všech knihoven a vytvoření instance PhysicsSDK. Celý kód je uveden ve výpisu 1.

```
#include <stdio.h>
#include "NxPhysics.h"
NxPhysicsSDK *sdk;
void main(int argc, char** argv)
{
    sdk = NxCreatePhysicsSDK(NX_PHYSICS_SDK_VERSION);
    sdk->setParameter(NX_SKIN_WIDTH, 0.01);
    if (!sdk)
    {
        printf ("Chyba - Nelze vytvořit SDK.");
    };
    if  (sdk) sdk->release();
}
```

Výpis 1: První program

V druhém řádku importuji hlavičkový soubor potřebný pro spuštění programu. Dalším řádkem deklaruji ukazatel na instanci třídy *NxPhysicsSDK* a v metodě *main()* pomocí funkce *PhysXLoader::NxCreatePhysicsSDK()* ji vytvořím a předám hodnotu ukazatele do proměnné. Při vytváření se automaticky nastaví implicitní hodnoty parametrů, které lze později změnit zavoláním funkce *NxPhysicsSDK::setParameter()*. Nastavitelných parametrů je celá řada. Na závěr funkce *NxPhysicsSDK::release()* uvolňuje vytvořený objekt z paměti.

Objekt typu *NxPhysicsSDK* byl vytvořen návrhovým vzorem Singleton. To znamená, že další objekt tohoto typu v rámci jedné aplikace nelze vytvořit. Z právě vytvořeného objektu je možné vytvořit instanci scény zavoláním funkce *NxPhysicsSDK::createScene()*. Jako parametr se uvádí popisná třída *NxSceneDesc*.

Popisná třída je často uváděna jako povinný parametr při vytváření většiny objektů. Její deklarace, úprava obsahu a způsob použití při vytváření objektu scény je uveden v následujícím výpisu.

```
void CreateScene()
{
    NxSceneDesc sceneDesc;
    sceneDesc.gravity = NxVec3(0.0f, −9.8f, 0.0f);
    sceneDesc.simType = NX_SIMULATION_HW;
    scene = sdk−>createScene(sceneDesc);
    if (!scene)
    {
        sceneDesc.simType = NX_SIMULATION_SW;
        scene = sdk−>createScene(sceneDesc);
        if (!scene)
        {
            printf ("Nelze vytvořit scénu.");
        }
    }
}
```

Výpis 2: Vytvoření scény

Pokud nebude dodatečně upraven žádný parametr popisné třídy sceneDesc bude ponecháno základní nastavení, které se inicializuje při jejím vytvoření. V tomto výpisu má třída sceneDesc nastaven parametr gravitace zadaný pomocí vektoru *NxVec3*. V části rozhodovacích bloků definuji, aby simulace běžela s podporou grafické karty PhysX, pokud není k dispozici simulace poběží jen za pomoci procesoru. Vytvořená scéna je registrovaná v rámci objektu *sdk* a pro případnou pozdější manipulaci je hodnota ukazatele uchována v ukazateli *NxScene *scene*.

Do vytvořeného objektu scény lze přidat objekty materiálu *NxMaterial*, které se v pozdější fázi aplikují na objekty scény. Parametry materiálu se přenášejí pomocí instance třídy *NxMaterialDesc* viz výpis 3.

```
void CreateMaterial()
{
    NxMaterialDesc materialDesc;
    materialDesc. restitution  = 0.99f;
    materialDesc. staticFriction  = 0.5f;
    materialDesc.dynamicFriction = 0.5f;
    NxMaterialIndex material = scene−>createMaterial(materialDesc);
}
```

Výpis 3: Vytvoření materiálu

Použitý materiál ovlivňuje jak se zachovají tělesa při vzájemné kolizi nebo styku jejich ploch. Parametr *restitution* ovlivňuje, jakou silou se odrazí sražené objekty. Hodnota může nabývat intervalu $\langle 0, 1 \rangle$. Vysoká hodnota znamená velkou sílu odrazu malá hodnota naopak tuto sílu pohlcuje. Další typickou vlastností těles je drsnost povrchu, která ovlivňuje jejich vzájemné tření. PhysX k tomuto účelu používá dva parametry. Prvním z nich je *staticFriction*. Definuje, jakou třecí sílu má těleso v klidu. Třecí sílu pohybujícího

se tělesa lze nastavit pomocí *dynamicFriction*. Obory hodnot jsou shodné s parametrem odrazové síly.

Úspěšně vytvořenou scénu je obvyklé obsadit aktéry. Prvním umístěným aktérem bude rovina, na které se veškerý děj na scéně odehrává. Aktér tohoto typu je statický. To znamená, že pro něj neplatí gravitace scény a při kolizi s jiným tělesem nemění svou polohu. Než vytvoříme samotného aktéra je nutné zadat základní parametry. K tomu slouží objekty tříd *NxActorDesc*, *NxPlaneShapeDesc*. *NxActorDesc* je používaný pro všechny typy aktérů. Druhý z uvedených je nutné zvolit dle typu aktéra.

```
NxActor* createPlane()
{
    NxPlaneShapeDesc shapeDesc;
    NxActorDesc actorDesc;
    actorDesc.shapes.pushBack(&shapeDesc);
    scene->createActor(actorDesc);
    return scene->createActor(actorDesc);
}
```

Výpis 4: Vytvoření roviny

Nyní zbývá nastavenou simulaci spustit. Způsob provedení je uveden ve výpisu 5.

```
void simulateFrame()
{
    scene->simulate(1/60.0f);
    scene->flushStream();
    scene->fetchResults(NX_RIGID_BODY_FINISHED, true);
}
```

Výpis 5: Spuštění simulace

Funkce *NxScene::simulate()* provede simulaci v časovém intervalu nastaveném parametrem. A funkce *NxScene::flushStream()* a *NxScene::fetchResults()* vyprázdní frontu scény a vrátí výsledek simulace. Takto implementovaná funkce simuluje pouze jeden rámec. Aby vznikl dojem animace, musí se rámce tvořit plynule za sebou. K tomu se nabízí umístit vytvořenou funkci do cyklu ukončeného dle požadavků. Ku příkladu nekonečný cyklus.

```
while(true)
{
    simulateFrame();
}
```

Výpis 6: Simulace v cyklu

Graficky zjednodušeně znázorněno, probíhá simulace v cyklu jako na obrázku 11.

Obrázek 11: Průběh simulace

Při spuštění programu simulace běží, ale nelze vidět výsledky. Pro vizualizaci je nutné použít grafickou knihovnu, která bude průběh simulace prezentovat na výstupní periferii, nebo použít PhysX Visual Debugger. Způsob použití je popsán v následující kapitole.

5.4 PhysX Visual Debugger a jeho použití

Tento software je součástí vývojového nástroje PhysX SDK a umožňuje základní vizualizaci aplikací. Lze jej připojit k jakékoli aplikaci založené na technologii PhysX a umožňuje:

- Vizualizaci objektů umístěných na scéně.
- Základní manipulaci s pevnými tělesy.
- Zaznamenání a možnost uložení výsledku simulace.

Spojení s PhysX aplikací je realizováno pomocí TCP/IP protokolu. Díky tomu je možné zobrazovat simulaci aplikace na místním i vzdáleném počítači, pokud jsou oba propojeni počítačovou sítí.

Před připojením k ladícímu softwaru je nutné znát IP adresu počítače, na kterém chceme program ladit. Způsob připojení demonstruje výpis 7.

```
NxPhysicsSDK *sdk;
void main(int argc, char** argv)
```

```
{
   sdk = NxCreatePhysicsSDK(NX_PHYSICS_SDK_VERSION);
   sdk->getFoundationSDK().getRemoteDebugger()->connect("localhost", 5425);

   ...

}
```

Výpis 7: Připojení k ladící aplikaci

V tomto případě se provede připojení na místní počítač na TCP portu 5425. V případě vzdáleného ladění se místo parametru „localhost“ uvede IP adresa vzdáleného počítače. Pokud se spojení nezdaří pokračuje simulace v činnosti, bez možnosti ladění.

5.5 Pevná tělesa

Pevné těleso je idealizace skutečných těles. Zavádí se proto, aby se při implementaci nemusela brát v úvahu deformace těles. Pro přesné vyjádření viz definice.

Definice 5.1 *Tuhé těleso je ideální těleso, jehož tvar ani objem se nemění účinkem libovolně velkých sil. Je charakterizováno hmotností a geometrickými rozměry, které vymezují určitý objem. Je tvořeno soustavou vzájemně pevně vázaných hmotných bodů [13].*

Při simulaci mohou nabývat dvou základních stavů.

- Statický – na objekt se vztahuje pouze detekce kolizí, při působení síly nedochází ke změně polohy objektu.
- Dynamický – mimo detekce kolizí na něj působí vnější síly, které mění jeho polohu a lze je spojovat pomocí potomků třídy *NxJoint*.

5.5.1 Tvary pevných těles

Každé pevné těleso je mimo jiné definováno svým tvarem, který se nemění pocelou dobu simulace. Z třídního diagramu na obrázku 5 je známo, že aktér se může skládat z jednoho či více tvarů třídy *NxShape*, nebo také z žádného. Pro tento účel je implementován parametr popisné třídy *NxActorDesc* typu *NxArray*, do kterého se jednotlivé položky ukládají. Nakonec aktéra vytvořím v rámci aktuální scény pomocí funkce *NxScene::createActor()*. Viz následující výpis.

```
//popisná třída tvaru krychle
NxBoxShapeDesc shapeDesc;
// vlastnosti aktéra
NxActorDesc actorDesc;
//vložení tvaru do popisné třídy aktéra
actorDesc.shapes.pushBack(&shapeDesc);
scene->createActor(actorDesc);
```

Výpis 8: Vytvoření aktéra

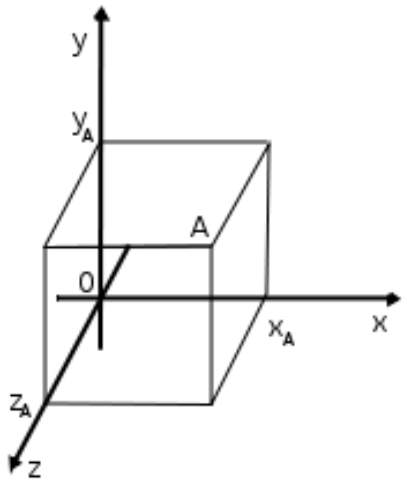
Jednotlivé tvary jsou definovány:

- krychle – určena hodnotou velikosti ve třech osách,
- koule – definována poloměrem,
- kapsle – k definici kapsle je zapotřebí určit:
  - výšku kapsle (válcová část),
  - poloměr krajních kulovitých ploch,
- konvexní síť – definována objektem typu *ConvexMesh*,
- trojúhelníková síť – určena objektem typu *TriangleMesh*,
- kolo – speciální typ tvaru popsáno v kapitole 5.5.4

5.5.2 Poloha, Orientace a těžiště těles

Objekt je mimo svého tvaru popsán svou polohou a orientací v prostoru. Poloha tělesa je údaj, vyjadřující umístění tělesa vzhledem ke vztažné soustavě. PhysX prostředí je založeno na 3D kartézské soustavě souřadnic, proto je pro určení polohy bodu zapotřebí tří souřadnic $A[x_A, y_A, z_A]$. Nejčastěji pomocí objektu typu *NxVec3*.

Poloha lze nastavit před vytvořením tělesa v objektu popisné třídy s jejímiž parametry se vytvoří. Pokud je těleso již vytvořeno lze polohu změnit pomocí funkce *NxActor::setGlobalPosition(NxVec3)*. Viz výpis 9.



Obrázek 12: Kartézská soustava souřadnic

Orientace tělesa je definována úhlem otočení tělesa vzhledem k osám souřadnic. Úhly otočení se definují pomocí matice 3×3 nebo quaterionu. Pro určení orientace pomocí

matic musíme mít pro každou osu x, y, z rotační matici. Vynásobením těchto tří matic nám vyjde konečná rotační matice. Quaternion má výhodu v tom, že všechny osy jsou obsaženy v jednom quaternionu.

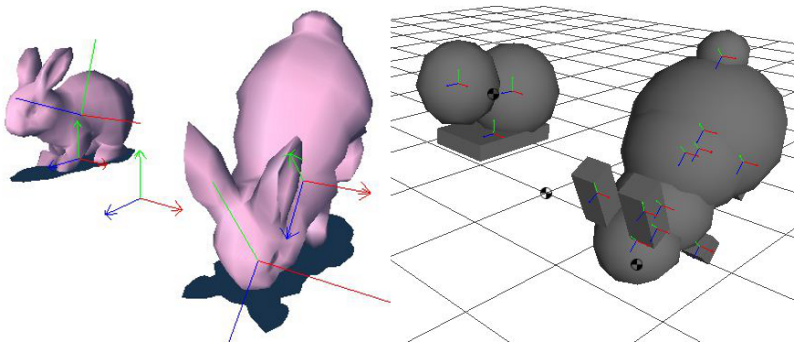
Definice 5.2 *Quaterniony jsou zobecněním komplexních čísel ve 3D prostoru. Zapisují se jako uspořádaná čtveřice* (w, x, y, z) *nebo v úspornějším formátu* $(w, \vec{v})$*, kde w je skalár a v je 3D vektor.*

Pro účel otočení kolem zvolené osy ve PhysX dosadíme za w hodnotu úhlu otočení a za $\vec{v}$ osy, ke kterým chceme rotaci provést. Ukázka práce s quaterniony se nachází ve výpisu 9.

V reálném světě, působí na každý bod tělesa v poli zemské tíže tíhová síla, která je úměrná hmotnosti daného bodu tělesa. Tato síla působí na bod (část tělesa) svisle dolů. Výslednice všech rovnoběžných tíhových sil udává celkovou tíhovou sílu $\vec{F_G}$ tělesa a leží na těžnici, což je spojnice těžiště tělesa a bodu závěsu. Otočením tělesa dojde ke změně polohy těžnice. Průsečík všech těžnic se nazývá těžiště tělesa [13].

Definice 5.3 *Těžiště pevného tělesa je působiště tíhové síly, která působí na těleso v homogenním tíhovém poli [13].*

Při vytváření tělesa ve PhysX se těžiště implicitně nastaví doprostřed tělesa. Způsob změny polohy těžiště je demonstrováno ve výpisu 9.



Obrázek 13: Těžiště tělesa

```
NxActorDesc actorDesc;
//natavení polohy tělesa v ˇpopisné třídě
actorDesc.globalPose.t = NxVec3(0,1,0);

//natavení orientace tělesa v ˇpopisné třídě
actorDesc.globalPose.M = NxQuat(45,NxVec3(0,1,0));

NxActor *cube = scene->createActor(actorDesc);

//nastavení polohy tělesa po jeho vytvoření
cube->setGlobalOrientation(NxQuat(0,NxVec3(0,1,0)));

//nastavení orientace tělesa po jeho vytvoření
```

```
cube->setGlobalPosition(NxVec3(0,5,0));

//změna těžiště tělesa po jeho vytvoření
cube->setCMassOffsetLocalPosition(NxVec3(1,1,1));
```

Výpis 9: Určení polohy orientace a těžiště aktéra

5.5.3 Působení síly a rychlost tělesa

Působením síly na těleso jej uvede z klidového stavu do pohybu. Každý pohyb se skládá ze dvou částí:

- Posuvný – při tomto pohybu se všechny body tělesa pohybují stejnou rychlostí po vzájemně rovnoběžných trajektoriích.
- Otáčivý pohyb – je pohyb, při němž se všechny body tělesa pohybují se stejnou úhlovou rychlostí po soustředných kružnicích, jejichž středy leží na ose otáčení. Otáčivý pohyb se děje vždy kolem nějaké okamžité osy otáčení.

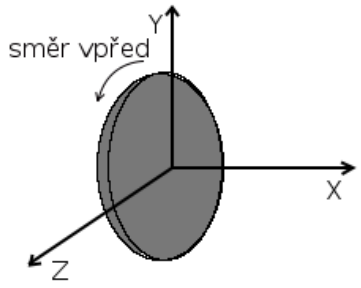
PhysX rozlišuje tyto dvě složky pohybu a umožňuje je nastavit zvlášť. Toho lze docílit funkcemi *NxActor::addForce()* pro posuvnou sílu a *NxActor::addTorque()* pro sílu otáčivou. Parametrem jsou vektory typu *NxVec3* udávající velikost síly pohybu (otáčení) v jednotlivých složkách kartézské soustavy souřadnic. Tyto funkce působí silou na střed tělesa. Pokud je žádoucí umístit působiště síly na jinou část tělesa využívá se funkce např. *NxActor::addForceAtPos()*.

```
NxActorDesc actorDesc;
NxActor *cube = scene->createActor(actorDesc);
//nastavení posuvné síly
cube->addForce(NxVec3(0,10000,0));
//nastavení otáčivé síly
cube->addTorque(NxVec3(0,10000,0));
//nastavení posuvné síly na určitou pozici
cube->addForceAtPos(NxVec3(0,0,10000),NxVec3(0,0,0));
```

Výpis 10: Působení síly na pevné těleso

5.5.4 Kolo

Kolo je speciální typ útvaru sloužící pro simulaci vozidel různých druhů. Způsob vytváření a připojení k aktérovi je stejný jako u jiných útvaru. Základní reprezentace je zobrazena na obrázku 14.



Obrázek 14: Reprezentace kola ve PhysX

Útvar má implementovánu sadu metod pro snadné nastavení vlastností. Tyto vlastnosti lze libovolně měnit v průběhu činnosti aplikace.

- *motorTorque* – udává velikost točivého momentu kola v Z+,
- *brakeTorque* – opět velikost točivého momentu, ale v opačném směru Z-,
- *steerAngle* – úhel otočení kola v ose Y v kladném, či záporném směru.

Na závěr následuje jednoduchý příklad vytvoření.

```
NxWheelShapeDesc wheelDescFL;
wheelDescFL.radius = 0.1f;
wheelDescFL.motorTorque = 200.0f;
wheelDescFL.brakeTorque = 0.0f;
wheelDescFL.steerAngle = 0.0f;
WheelDescFL.localPose.t = NxVec3(0.0f,0.1f,0.0f);
actorDesc.shapes.pushBack(&wheelDescFL);
```

Výpis 11: Příklad vytvoření kola

5.5.5 Spínač událostí

Spínač je speciální vlastnost tvaru (*NxShape*), kdy daným tvarem mohou volně procházet jiná pevná tělesa, avšak při průchodu se vyvolá událost, která je dále zpracovávána obsluhující třídou. Tvar se při vytváření může stát spínačem umístěním konstanty *NX_TRIGGER_ENABLE* do vlastnosti popisné třídy *NxBoxShapeDesc.shapeFlags*.

Spínač by sám o sobě neměl smysl, pokud by neexistovala obslužná funkce, která by jeho události zpracovávala. K tomuto účelu je implementována třída *NxUserTriggerReport* z níž dědí výsledná obslužná třída. Příklad vytvoření prezentuje výpis 12.

```
class Trigger : public NxUserTriggerReport
{
   void onTrigger(NxShape& sensorShape, NxShape& detectedShape, NxTriggerFlag status)
   {
```

```
        if (status & NX_TRIGGER_ON_ENTER)
        //obslužný kód pro vstup
        if (status & NX_TRIGGER_ON_LEAVE)
        //obslužný kód při opuštění
    }
}Trigger;
```

Výpis 12: Příklad obslužné třídy spínače

Jedinou metodou třídy je metoda *onTrigger* jejíž parametry jsou:

- tvar spínače, který byl narušen,
- tvar , který spínač narušil,
- typ události.

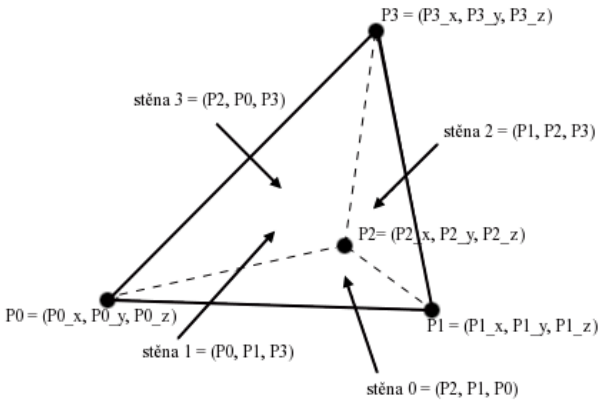
Pokud je znám tvar, lze pomocí funkce *NxShape::getActor()* získat referenci na rodičovského aktéra. To nabízí širokou škálu použití spínačů.

5.5.6 Troúhelníkové sítě

Trojúhelníkové sítě je nástroj na vytváření komplexních tvarů objektu složených z dílčích trojúhelníků. Ve své práci používám následující dva typy:

- Convex Mesh (konvexní síť) – vhodné pro vytváření složitějších statických i dynamických objektů, kdy skládání objektu z primitivních tvarů by bylo příliš obtížné.
- Triangle Mesh (trojúhelníková síť) – vhodná pro vytváření rozsáhlých statických objektů např. terénů.

Trojúhelníkové sítě mohou nabývat jakéhokoli tvaru. Tím vzniká problém, protože systém detekce kolizí uživatelsky definovaný tvar tělesa nezná, a proto kolize objektů neprobíhá správně nebo vůbec. K řešení tohoto problému se využívá instance třídy *NxCookingInterface*, která uživatelský objekt převede do vhodného tvaru pro simulaci. Před převedením je nutné správně definovat požadovaný tvar objektu. Jako příklad uvedu následující obrázek.



Obrázek 15: Příklad tělesa z dílčích trojúhelníků

Těleso je tvořeno čtyřmi vrcholy P_0, P_1, P_2, P_3 z nichž každý je definován svou polohou v prostoru $P_0 = (x_0, y_0, z_0) \ldots$ Počet a poloha bodů jsou prvními parametry. Plášť tělesa je tvořen čtyrmi stěnami trojúhelníkového tvaru a každá plocha stěny je určena třemi body viz obrázek. Počet ploch a výčet bodů, které tyto stěny tvoří jsou druhou sadou parametrů. Počty ploch a bodů uložím do proměnných číselného typu a výčet bodů a trojúhelníků do pole prvků.

```
NxI32 pocetBodu = 4;
NxI32 pocetTrojuhelniku = 4;

NxVec3 poleBodu[4] =
{NxVec3(p0_x, p0_y, p0_z),
 NxVec3(p1_x, p1_y, p1_z),
 NxVec3(p2_x, p2_y, p2_z),
 NxVec3(p3_x, p3_y, p3_z)};
NxU16 poleTrojuhelniku[12] =
{2,1,0,
 0,1,3,
 1,2,3,
 2,0,3};

static NxCookingInterface *cooking = NxGetCookingLib(NX_PHYSICS_SDK_VERSION);
cooking->NxInitCooking();

NxTriangleMeshDesc triangleMeshDesc;
triangleMeshDesc.numVertices = pocetBodu;
triangleMeshDesc.numTriangles = pocetTrojuhelniku;
triangleMeshDesc.pointStrideBytes = sizeof(NxVec3);
triangleMeshDesc.triangleStrideBytes = 3*sizeof(NxU32);
triangleMeshDesc.points = poleBodu;
triangleMeshDesc.triangles = poleTrojuhelniku;

//uložení do souboru
cooking->NxCookTriangleMesh(triangleMeshDesc, UserStream("c:\\triangleMesh.bin", false));
```

```
//načtení ze souboru a vytvoření objektu
NxTriangleMesh triangleMesh = physicsSDK->createTriangleMesh(UserStream("c:\\triangleMesh.
    bin", true));
```

Výpis 13: Vytvoření trojúhelníkové sítě

V příkladu se data sítě nejprve uloží do souboru a poté se pomocí tohoto souboru vytvoří objekt typu*NxTriangleMesh* (v případě konvexního tělesa je to objekt typu *NxConvexMesh*). Druhou možností je uložení výstupního proudu dat do paměti. To by znamenalo vytvářet data sítě po každém spuštění simulace a zpomalit tak běh programu. Po přípravě dat sítě následuje vytvoření reálného aktéra. I zde je nutné nejprve před vytvořením tělesa vyplnit objekty popisných tříd. Viz výpis 13.

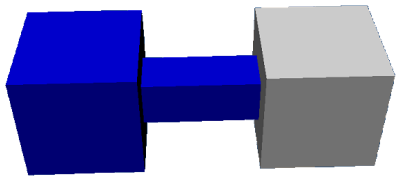
```
NxTriangleMeshShapeDesc triangleShapeDesc;

//data sítě
triangleShapeDesc.meshData= triangleMesh;
NxBodyDesc bodyDesc;
NxActorDesc actorDesc;
actorDesc.shapes.pushBack(&triangleShapeDesc);
actorDesc.body= &bodyDesc;
actorDesc.density= 1.0f;
NxActor *newActor = scene->createActor(actorDesc);
```

Výpis 14: Aplikace sítě na aktéra

5.5.7 Spoje pevných těles

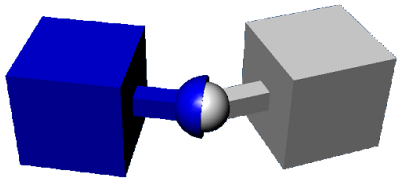
Spoje umožňují spojení dvou těles za účelem vytvoření jednoduchých i složitějších mechanických seskupení. Analogicky jako např. v lidské kostře se nachází různé druhy kloubů, i PhysX implementuje sadu spojů, které se od sebe navzájem liší omezením vzájemného pohybu těles. PhysX implementuje následující typy spojů:

- Pevný (*NxFixedJoint*) – vytváří pevné spojení mezi tělesy a neumožňuje žádný pohyb ve všech osách.



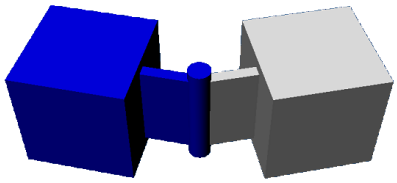
Obrázek 16: Pevný spoj [8]

- Kulovitý (*NxSphericalJoint*) – spoj tvořen dvěma styčnými kulovitými plochami s nastavitelnou hybností ve všech osách. Vhodný pro použití simulace lan či některých kloubů (kyčelní).



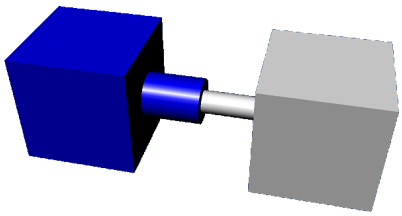
Obrázek 17: Kulovitý spoj [8]

- Otočný (*NxRevoluteJoint*) – umožňuje omezený, otáčivý pohyb pouze v jediné ose. Další dvě osy jsou fixovány. Použití například pro dveřní panty, kolenní kloub apod.



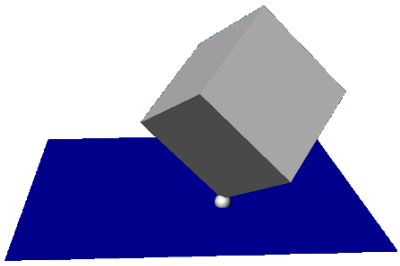
Obrázek 18: Otočný spoj [8]

- Válcový (*NxCylindrialJoint*) – simuluje posuvný a otáčivý pohyb v jedné ose. Ostatní osy jsou fixovány. Typickým příkladem je píst v pístnici.



Obrázek 19: Válcový spoj [8]

- Bod na rovině (NxPointInPlane) – umožňuje pohyb bodu tělesa pouze po vybrané ploše druhého tělesa.



Obrázek 20: Spoj Bodu a roviny [8]

Konkrétní typy spojů se vytvářejí v rámci scény pomocí funkce *NxScene::createJoint()*. Typ spoje se určuje dle typu třídy. Při vytváření je nutné uvést jako parametr popisnou třídu daného spoje. Vzhledem k velkému počtu typů spojů uvedu pouze jeden demonstrativní příklad vytvoření pevného spoje mezi dvěma aktéry.

```
NxFixedJointDesc fixedDesc;
fixedDesc.actor[0] = actor0;
fixedDesc.actor[1] = actor1;
NxFixedJoint *fixedJoint=(NxFixedJoint*)gScene->createJoint(fixedDesc);
```

Výpis 15: Vytvoření pevného spoje

5.6 Silové pole

Silová pole jsou objekty, které působí zvolenou silou na aktéry, jež se vyskytnou v jejich působišti. Ovlivněnými tělesy mohou být dynamická pevná tělesa, tkaniny, tekutiny, měkká tělesa. Silová pole mohou nabývat tvaru:

- krychle,
- koule,
- kapsle
- konvexní síť

Stejně jako u aktérů, pole může být tvořeno více tvary. Přesto objem ani povrch pole není hmotný. Tělesa tedy mohou pronikat dovnitř tělesa i ven. Při vytváření konečného objektu typu *NxForceField* je nutné do jeho popisné třídy uvést informace o velikosti, tvaru a poloze silového pole a jeho působící síle. K tomu slouží objekt *NxForceFieldLinearKernel* a popisná třída *NxBoxForceFieldShapeDesc*. Příklad vytvoření se nachází v následujícím výpisu.

```
NxForceFieldLinearKernelDesc lKernelDesc;
lKernelDesc.constant = NxVec3(1,1,1); //velikost působící síly
NxForceFieldLinearKernel* lKernel; //objekt uchovávající mj. velikost působící síly
lKernel = scene->createForceFieldLinearKernel(lKernelDesc);
```

```
NxBoxForceFieldShapeDesc boxField; //tvar pole je krychle
boxField.dimensions = NxVec3(15,15,15); //velikost pole
boxField.pose.t = NxVec3(0,10,0);

NxForceFieldDesc fieldDesc; //popisná třída silového pole
fieldDesc.coordinates = NX_FFC_CYLINDRICAL;
fieldDesc.includeGroupShapes.pushBack(&boxField);
fieldDesc.kernel = lKernel;

NxForceField* ffield = scene->createForceField(fieldDesc);
```

Výpis 16: Vytvoření silového pole

Příklad použití tohoto objektu je kupříkladu vítr, lokální stav beztíže a další.

5.7 Tekutiny

PhysX umožňuje simulaci kapalin a plynů pomocí systému částic v reálném čase. Na scéně mohou být vytvořeny samostatně, nebo mohou vytékat z vybraných objektů.

5.7.1 Vytvoření tekutin

Pro vytvoření tekutiny vlastní třída *NxScene* funkci *createFluid()* s použitím popisné třídy *NxFluidDesc* jako parametr. Nyní popíši jak pomoci této třídy specifikovat fyzikální vlastnosti kapalin.

- *maxParticies* – maximální počet částic. Po překročení tohoto limitu se nebudou vytvářet další částice.
- *kernelRadiusMultiplier* – ovlivňuje poloměr vlivu pro každou částici.
- *restDensity* – hustota kapaliny
- *restParticlesPerMeter* – parametr udávající množství částic v tekutině o objemu jednoho metru krychlovém. Spolu s předchozími dvěma parametry slouží pro výpočet hmotnosti kapaliny m a poloměru vlivu r.

$$m\,[kg] = \frac{restDensity\,[kg * m^{-3}]}{restParticlesPerMeter^3\,[m^{-1}]}$$

$$r\,[m] = \frac{kernelRadiusMultiplier}{restParticlesPerMeter\,[m^{-1}]}$$

- *stiffness* – tuhost kapaliny ovlivňuje míru stlačitelnosti.
- *viscosity* – viskozita kapaliny.
- *damping* – tlumící parametr. Používá se pro ztlumení rychlosti částic.

- *externalAcceleration* – externí zrychlení aplikovaná na částice. Umožňuje simulaci např. větru. Udáváno v $[m * s^{-2}]$.

K vytvoření již jen zbývá definovat samotné částice. Ty jsou určeny svou polohou. Jako řešení se vybízí použít pole prvků typu *NxVec3*. Způsob implementace demonstruje výpis 17.

```
NxI32 particleNum = 0;
NxVec3 particleBuffer[30000];

NxParticleData particles;
particles.numParticlesPtr = &particleNum;
particles.bufferPos = &particleBuffer[0].x;
particles.bufferPosByteStride = sizeof(NxVec3);

// Vytvoření popisné třídy
NxFluidDesc fluidDesc;

...

fluidDesc.initialParticleData = particles;
fluidDesc.particlesWriteData = particles;

fluid = scene->createFluid(fluidDesc);
```

Výpis 17: Vytvoření kapaliny

5.7.2 Přítok a odtok tekutin

V PhysX jsou implementovány nástroje pro simulace kupříkladu fontán nebo vodovodního kohoutku s odtokem, kdy kapalina vytéká z objektu a následně se ztrácí. Výtok kapalin se vytváří pomocí funkce *NxFluid::createEmitter*. K předání argumentů slouží opět popisná třída tentokrát typu *NxFluidEmitterDesc*. K upřesnění požadavků lze nastavit tyto argumenty:

- *relPose* – relativní pozice emitoru vzhledem k objektu, ze kterého kapalina vytéká.
- *type* – volba zda-li bude kapalina tryskat (fontána) nebo volně vytékat (vodovodní kohoutek).
- *shape* – tvar průřezu proudu částic.
- *particleLifetime* – existence částic udávaná v sekundách $[s]$. Pokud je udána nulová hodnota, částice zmizí pouze odtokem.
- *maxParticles* – maximální počet emitovaných částic. Pokud počet vypuštěných částic dosáhne tohoto limitu, tok částic se zastaví do doby, než počet částic pod limit opět neklesne.
- *fluidVelocityMagnitude* – počáteční rychlost emitovaných částic.

- *rate* – množství emitovaných částic za sekundu.
- *dimensionX, dimensionY* – velikost výtokové plochy.
- *randomAngle* – náhodná úhlová odchylka směru emitovaných částic od směru emitace.
- *randomPos* – rozptyl částic
- *frameShape* – parametr typu *NxShape* udává, ze kterého objektu má kapalina vytékat.

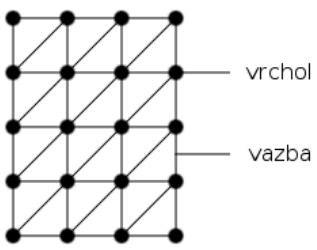
K odtoku kapaliny slouží objekty z kategorie pevných těles. Pokud se takto vytvořeného tělesa dotkne částice kapaliny, bude odstraněna z paměti. Způsob vytvoření je uveden ve výpisu.

```
//vytvoření přítoku
NxFluidEmitterDesc emitterDesc;
emitterDesc.setToDefault();

//nastavení vlastností

fluidEmitter = fluid->createEmitter(emitterDesc);

//vytvoření odtoku
NxBoxShapeDesc shapeDesc;
shapeDesc.setToDefault();
//konstanta označující odtokový objekt
shapeDesc.shapeFlags|=NX_SF_FLUID_DRAIN;
NxActorDesc actorDesc;
actorDesc.shapes.pushBack(&shapeDesc);
NxActor *drainActor=scene->createActor(actorDesc);
```

Výpis 18: Vytvoření přítoku a odtoku kapaliny

5.8 Tkaniny

Simulace textilu a jiných tkaných materiálu je dalším nástrojem PhysX enginu. Umožňuje simulaci chování textílií, ale i pevných těles, které lze do určité míry deformovat. Principem implementace je síť bodů spojená vazbami do dílčích trojúhelníků. Viz obrázek 21.



Obrázek 21: Struktura tkaniny ve PhysX

Síť bodů může nabývat jakýkoli tvar. To znamená velkou univerzálnost použití od simulace jednoduchých pláten, kusy oblečení, papíru, ale i pevných těles (plastové láhve). Na rozdíl od objektu typu *NxTriagleMesh* je dynamická a není omezena množstvím ploch jako objekt typu *NxConvexMesh*, tudíž se hodí pro vytváření složitějších a detailnějších tvarů.

5.8.1 Vytvoření tkanin

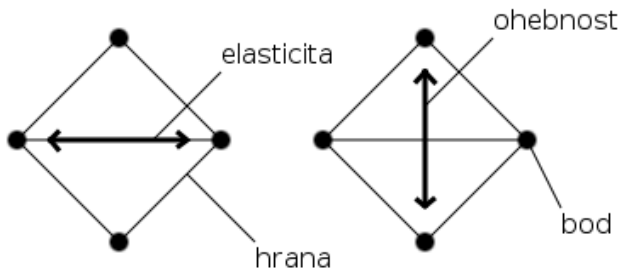
Podobně jako u vytváření objektů typu *NxTriangleActor* i zde je nutné připravit síť bodů. K tomu použiji funkci *NxCookingInterface::NxCookClothMesh()* a *NxPhysicsSDK::createClothMesh()*. Provedení je analogické jako z kapitoly 5.5.6.

```
NxInitCooking();
NxClothMeshDesc meshDesc;

...

MemoryWriteBuffer wb;
if (!NxCookClothMesh(meshDesc, wb))
   return false;
MemoryReadBuffer rb(wb.data);
clothMesh = scene->getPhysicsSDK().createClothMesh(rb);
```

Výpis 19: Vytvoření sítě bodů

Po vytvoření dat sítě lze přistoupit k vytvoření objektu typu *NxCloth*. K tomu je opět zapotřebí objekt popisné třídy (*NxClothDesc*). Mezi nejdůležitější parametry této třídy patří:

- *Stretching Stiffness* (elasticita) – definuje, do jaké míry jsou natažitelné vazby mezi body.
- *Bending Stiffness* (ohebnost) – omezení úhlu ohybu mezi protilehlými body. Vysokou ohebnost má kupříkladu bavlna, naopak nízkou má třeba papír.



Obrázek 22: Elasticita a ohebnost tkanin

- *Density* (hustota) – nepřímo udává hmotnost tkaniny. Hmotnost m je určena vzorcem:
$$m\,[kg] = \frac{1}{3} * triangleArea\left[m^3\right] * density\left[kg * m^{-3}\right]$$
- *Thickness* (tloušťka) – tloušťka plátna.
- *Friction* (tření) – založeno na stejném principu jako u pevných těles.

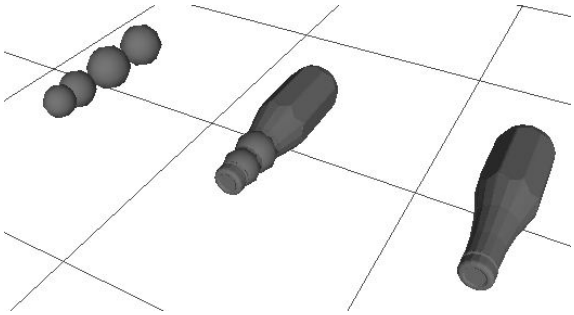
Objekt třídy *NxCloth* se vytváří pomocí funkce *NxScene::createCloth()* s parametrem typu *NxClothDesc* viz výpis 20.

```
NxClothDesc clothDesc;

...

NxMeshData receiveBuffers;
clothDesc.clothMesh = clothMesh;
clothDesc.meshData = receiveBuffers;
cloth  = scene->createCloth(clothDesc);
```

Výpis 20: Vytvoření tkaniny

5.8.2 Další vlastnosti tkanin

5.8.2.1 Vzájemná kolize V současné verzi nemají objekty tohoto typu implementovány detekce kolizí s jinými tkaninami, nýbrž kolidují pouze se sebou samými. To může být problém při simulaci více objektů tohoto typu. Objekty do sebe nebudou narážet, ale vzájemně sebou proniknou. PhysX tento problém řeší umístěním neviditelného pevného tělesa doprostřed objektu pomocí funkce *NxCloth::attachToCore()*. Tím se zajistí vzájemná kolize a navíc se plášť síťových bodů bude při kolizi deformovat.



Obrázek 23: Umístění pevných těles do jádra tkanin

5.8.2.2 Trhání tkanin Jak již bylo zmíněno, tkaniny se skládají z vrcholů a vazeb (obrázek 21), které tvoří dílčí trojúhelníky plochy. PhysX umožňuje vazbám mezi body nastavit meze natažitelnosti. Po překročení této meze dojde k trvalému přetržení vazby.

K umožnění přetržení je nutné nastavit popisné třídě konstantu *NxClothDesc.flags = NX_CLF_TEARABLE* a reálné číslo prahu natažitelnosti *NxClothDesc.tearFactor*.

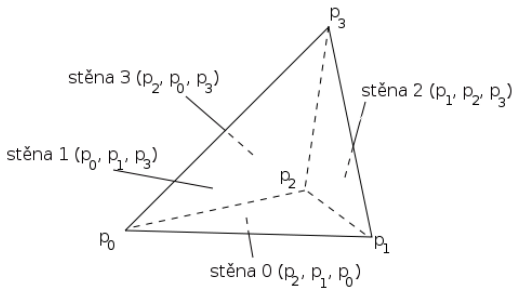
5.8.2.3 Vnitřní tlak tkanin Slouží pro vytvoření vnitřního tlaku uzavřených těles tohoto typu. Používá se pro simulaci nafouknutých objektů (létající balón). Tato funkce se aktivuje nastavením konstanty v popisné třídě *NxClothDesc.flags = NX_CLF_PRESSURE* a reálné číslo velikosti tlaku *NxClothDesc.pressure*.

5.9 Měkká tělesa

PhysX umožňuje simulaci měkkých těles, která podléhají deformaci, ale po ukončení působení deformační síly se těleso vrátí do původního stavu. V praxi lze objekty tohoto typu použít pro simulaci např. pneumatiky vozidel, lidského těla nebo nafukovacího míče.

5.9.1 Vytvoření měkkého tělesa

Na rozdíl od předchozích typů objektů není tvořen trojúhelníkovou sítí, ale obecným čtyřstěnem (viz obrázek 24), tedy trojrozměrným tělesem se čtyřmi vrcholy a čtyřmi trojúhelníkovými stěnami. Výsledný objekt tedy není obal složený ze sítě trojúhelníků, ale má hmotný objem. Tento typ objektu není vhodný pro vizualizace, používá se jako vnitřní simulační vrstva, na kterou se posléze aplikuje vnější vrstva.



Obrázek 24: Obecný čtyřstěn

Vytváření je podobné jako u tkanin. Nejprve je nutné vytvořit síť čtyřstěnů a provést její převedení. Pro zadání jednotlivých bodů a stěn sítě slouží objekt popisné třídy *NxSoftBodyMeshDesc* a převedením pomocí funkce *NxCookingInterface::NxCookSoftBodyMesh()* lze vytvořit požadovanou síť typu *NxSoftBodyMesh*.

Poté lze tuto převedenou síť použít jako jeden z parametrů popisné třídy *NxSoftBodyDesc*, která je nutná pro vytvoření konečného objektu typu *NxSoftBody*. Viz zjednodušený výpis 21.

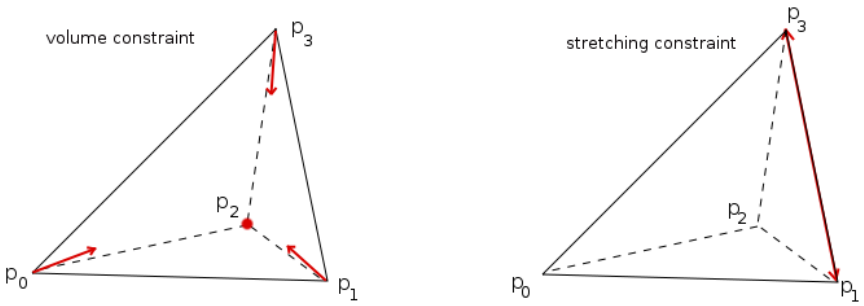
```
NxInitCooking();
NxSoftBodyMeshDesc meshDesc;
```

```
...

MemoryWriteBuffer wb;
if (!NxCookSoftBodyMesh(meshDesc, wb))
    return false;
MemoryReadBuffer rb(wb.data);
softBodyMesh = scene->getPhysicsSDK().createSoftBodyMesh(rb);

NxSoftBodyDesc softBodydesc;
NxMeshData receiveBuffers;

softBodydesc.softBodyMesh = softBodyMesh;
softBodydesc.meshData = receiveBuffers;

softBody = scene->createSoftBody(softBodydesc);
```

Výpis 21: Vytvoření měkkého tělesa

Při vytváření měkkých těles lze v rámci popisné třídy *NxSoftBodyDesc* nastavit následují parametry umožňující přiblížit tvořený objekt reálnému tělesu.

- *Volume Stiffness* (objemová pružnost) – určuje tuhost deformace objemu jednotlivých čtyřstěnů. Viz obrázek 25 vpravo.
- *Stretching Stiffness* (pružnost při tahu) – ovlivňuje tuhost deformace hran jednotlivých čtyřstěnů. Viz obrázek 25 vlevo.



Obrázek 25: Rozdíly mezi pružnostmi měkkých těles

- *Density* (hustota) – nepřímo udává hmotnost měkkého tělesa. Hmotnost m je určena vzorcem:

$$m\,[kg] = \frac{1}{4} * objem\left[m^3\right] * hustota\left[kg * m^{-3}\right]$$

- *Particle Radius*

5.9.2 Další vlastnosti měkkých těles

5.9.2.1 Vzájemná kolize Stejně jako v případě tkanin, ani měkká tělesa zatím nemají implementovanou vzájemnou kolizi. Řešení je stejné jako u tkanin tedy připevnění pevného tělesa do jádra objektu. Pro tento účel je implementována sada funkcí s různými možnostmi připojení. Například připojení ke kolidujícím objektům při počátku simulace, připojení ke konkrétnímu aktéru a další.

5.9.2.2 Trhání měkkých těles I zde platí analogie jako u tkanin. Dílčí čtyřstěny tvořící výsledné těleso se mohou oddělit, pokud je působící síla větší než soudržná síla vazeb. Tuto vlastnost lze nastavit v popisné třídě parametrem
NxSoftbodyDesc.flags = NX_SDF_TEARABLE a reálné číslo prahu pevnosti vazeb *NxSoftbodyDesc.tearFactor*.

6 Implementace příkladů

6.1 Knihovna OpenGL

V kapitole 5.4 jsem se zmínil o možnostech použití vizuálního ladícího softwaru. Avšak pro vizualizaci reálných aplikací je jeho použití zcela nevhodné. Pro účely zobrazit danou aplikaci je zapotřebí použití grafické knihovny. Pro účely mé práce jsem zvolil grafickou knihovnu OpenGL, která je pro tyto účely hojně používána.

„Knihovna OpenGL je průmyslový standart specifikující multiplatformní rozhraní pro tvorbu aplikací počítačové grafiky různého využití .“[14]

Principem zobrazování pohybujícího se objektu je rychlý sled snímků. Na každém z nich je zobrazen dílčí pohyb objektu. Celkový sled snímků tvoří kompletní animaci pohybu. Na tomto principu pracuje i OpenGL. Spuštění OpenGL aplikace probíhá v následujících fázích:

1. Vytvoření okna v rámci používaného operačního systému.
2. Inicializace OpenGL.
3. Nastavení souřadnicového systému.
4. iterační nekonečný cyklus v jehož rámci se provádějí akce jako:
   - Kontrola stištění kláves, myši.
   - Překreslení okna.

Z programátorského hlediska se OpenGL chová jako stavový automat. Díky tomu je možné během vykreslování průběžně měnit vlastnosti vykreslovaných primitiv (barva, průhlednost) nebo celé scény (volba způsobu vykreslování, transformace) zadáním konkrétních funkcí a toto nastavení zůstane zachováno do té doby, než bude změněno. Vykreslování scény se provádí voláním funkcí OpenGL, kdy se vykreslí výsledný rastrový obrázek. Tento rastrový obrázek je uložen v tzv. framebufferu, kde je každému pixelu přiřazena barva, hloubka, alfa složka a další atributy. Z framebufferu lze získat pouze barevnou informaci a tu je možné následně zobrazit na obrazovce.

Obrázek 26: Princip vykreslování

```
void onResize(int vyska, int  sirka) {..}
void onDisplay(void){..}
void onKeyboard(unsigned char key, int x, int y){...}
void onIdle(void){..}
```

```
int main(int argc, char **argv)
{
 glutInit (&argc, argv); // inicializace knihovny GLUT
glutCreateWindow("Priklad"); // vytvoření okna pro kreslení
glutReshapeWindow(200, 200); // změna velikosti okna
glutPositionWindow(100, 100); // změna pozice levého horního rohu
glutDisplayFunc(onDisplay); // registrace funkce volané pri překleslování okna
glutReshapeFunc(onResize); // registrace funkce při změně velikosti okna
glutKeyboardFunc(onKeyboard); // registrace funkce při stlačení ASCII klávesy
glutIdleFunc(onIdle); // registrace funkce volané při volném časovém slotu
glutMainLoop(); // nekonečná smyčka pro volání funkcí
return 0;
}
```

Výpis 22: Základní OpenGL aplikace

6.2 Modelování a import objektů

Pro modelování složitějších objektů ve všech následujících příkladech používám 3D modelovací software Blender. Zhotovené objekty exportuji jako soubor formátu Wavefront, který následně importuji do své PhysX aplikace.

Wavefront je standardní nekomprimovaný formát pro reprezentaci polygonálních dat v ASCII formě (pomocí . *Obj* přípony souboru). Využívá se pro ukládání geometrických objektů tvořených čarami, polygony, křivkami či plochami. Čáry a polygony jsou popsány pomocí bodů, křivky a plochy pomocí kontrolních bodů a dalších informací podle typu křivky. [16]

Jak jsem již zmínil formát souboru je nekomprimovaný, tudíž poměrně snadno čitelný. Nevyžaduje ani žádné speciální záhlaví. Každý řádek má svou klíčovou značku, za kterou následuje informace o dané entitě. Nejpoužívanější značky jsou:

- *v* – Geometrický vrchol
- *vn* – Normálový vektor
- *p* – Bod
- *f* – Ploška
- *usemtl* – Název materiálu
- *mtlib* – Materiál knihovny

Namodelované objekty jsem exportoval jako soubor polygonů. Příklad jednoduchého polygonu:

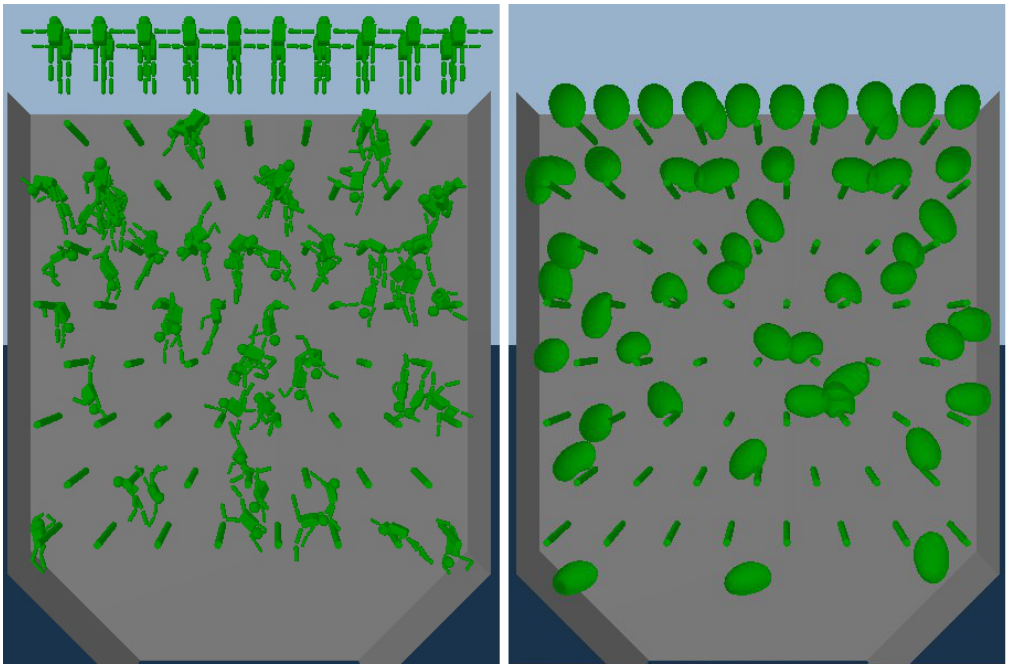
```
v 0,0 0,0 0,0
v 0,0 1,0 0,0
v 1,0 0,0 0,0
f 1 2 3
```

Výpis 23: Příklad polygonu

Za klíčovým znakem v následují souřadnice bodu v jednotlivých osách kartézské soustavě souřadnic. Za klíčovým znakem f je zapsáno pořadí bodů tvořící polygon. V tomto případě je polygon tvořen třemi vrcholy o souřadnicích $A = [0, 0, 0]$, $B = [0, 1, 0]$, $C = [1, 0, 0]$.

6.3 Příklad Collision Test

První příklad je primárně určen na prověřování kolizí objektů. Principem je vytváření většího množství dynamických objektů jednoho typu (pevné, měkké těleso, těleso z tkaniny) najednou a při pádu simuluji jejich kolize se statickými pevnými tělesy. Na scénu v průběhu času (interval lze nastavit jako parametr v konfiguračním souboru), nebo ručně pomocí klávesnice přibývají stále další aktéři. Dle nastavení to mohou být objekty jednoho, nebo více typů. To zvyšuje množství kolizí a tím i nároky na výpočet (vhodné pro testování výkonu hardware).



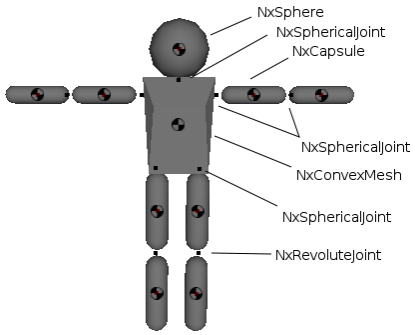
Obrázek 27: Příklad Collision Test

6.3.1 Vytvoření statických objektů

Základem aplikace je trychtýř s větším počtem vnitřních překážek. Trychtýř je statický objekt složený ze čtyř aktérů. Každý tvoří jednu stěnu objektu. Jako reprezentaci jsem zvolil objekt trojúhelníkové sítě (*NxTriangleMesh* popsán v kapitole 5.5.6). Jeho vnitřní překážky jsou rovněž statické, ale jako typ objektu jsem zvolil pevné konvexní těleso (*NxConvexMesh*) složené z jednotlivých překážek.

6.3.2 Vytvoření loutky

Pro demonstraci kolizí pevných těles jsem vytvořil třídu aktérů a spojů reprezentující dřevěnou loutku. Loutka je složená z částí a spojů zobrazeném na obrázku 28.



Obrázek 28: Schéma loutky

Trup byl namodelován v aplikaci Blender zatímco útvary ostatních části je možné vytvořit ve PhysX přímo (koule, kapsle). Spoje byly použity většinou kulovité, s výjimkou kolenních kloubů kde byly použity spoje otočné.

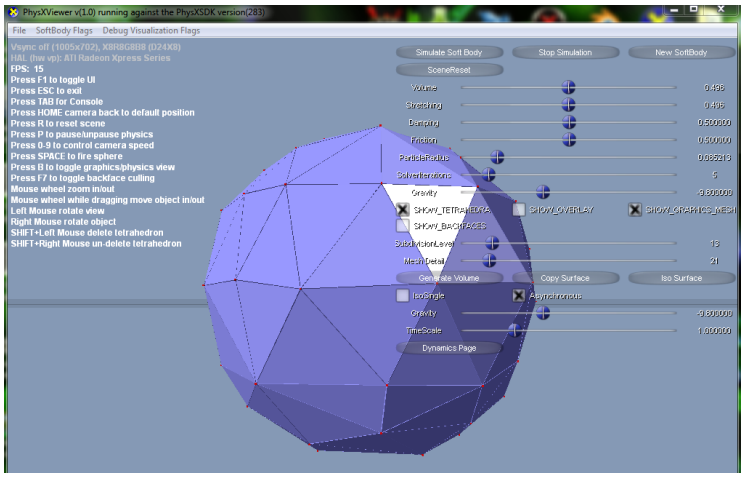
6.3.3 Vytvoření míče

Míč vytvořený z objektu tkaniny *NxCloth* lze pomocí konfiguračního souboru nastavit jako roztržitelný či nikoliv. Model jsem opět importoval jako Wavefront objekt a princip vytvoření je shodný jako v kapitole 5.8.1.

6.3.4 Vytvoření měkkého tělesa

V kapitole 5.9.1 jsem se zmiňoval, že tělesa tohoto typu jsou tvořena obecným čtyřstěnem. Vytvoření složitějších objektů pomocí modelování trojúhelníkové sítě či dokonce ručním zadáváním bodů by bylo velice pracné. K účelu vytváření sítě čtyřstěnů je nejvhodnější použít software PhysXViewer, který je obsažen v instalačním balíku PhysX SDK. Program umožňuje importovat několik formátů 3D modelování (Wavefront, COLLADA, XML).

Převedení lze provést v různých stupních kvality. Výstupním formátem je soubor typu *.tet*, který je velice podobný standardu Wavefront.

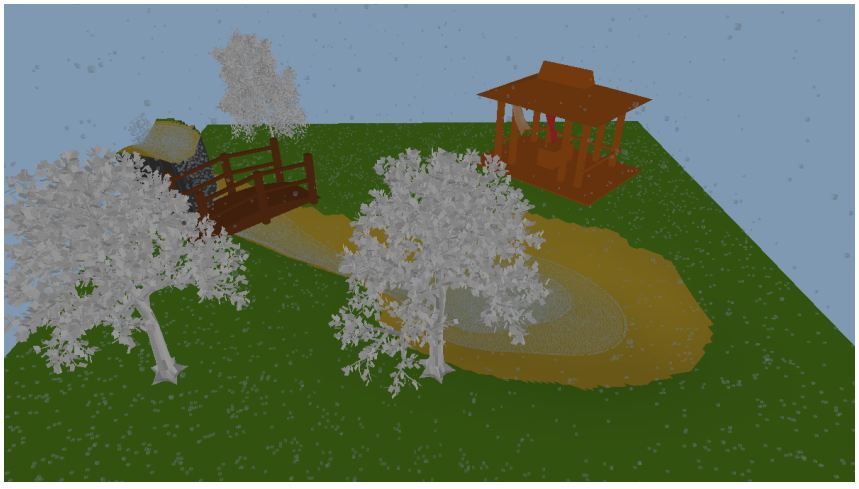


Obrázek 29: Ukázka aplikace PhysXViewer

Takto převedený objekt jsem importoval do své aplikace. Stejně jako u objektu z tkaniny i tento lze nastavit jako rozpadnutelný.

6.4 Příklad Japanese Garden

Další příklad reprezentuje komplexní řešení přírodního prostředí s přírodními vlivy. Celá scéna je zobrazena na obrázku 30. Na rozdíl od předchozích příkladů, ve kterých figurovaly objekty stejného typu, jsem se snažil využít co největší množství typů objektů a sladit je do jediné scény.



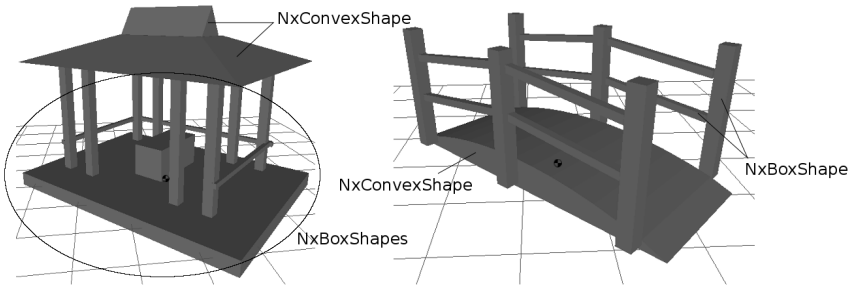
Obrázek 30: Příklad Japanese Garden

Základ scény tvoří terén složený z trojúhelníkových polygonů třídy *NxTriangleMesh*, který je vhodný pro tyto účely. Terén obsahuje tři druhy materiálů:

- Trávu – nízká hodnota odrazivosti (*restitution*) a vysoká hodnota statického (*staticFriction*) i dynamického tření (*dynamicFriction*).
- Bahno v korytě potůčku – nízká hodnota odrazivosti, nízká hodnota statického tření a ještě nižší hodnota tření dynamického (pokud objekt na blátě uklouzne, zastavuje obtížně).
- Skála u vodopádu – vyšší hodnota odrazivosti a střední až vyšší hodnota tření.

Stejným způsobem tedy pomocí trojúhelníkové sítě byla do scény vložena socha postavy s přiřazením materiálu skály.

Dalšími objekty jsou zahradní altánek a most. Oba jsou pevná tělesa typu *NxActor* složená z většího počtu částí (*NxShape*) různých tvarů. Viz obrázek 31.

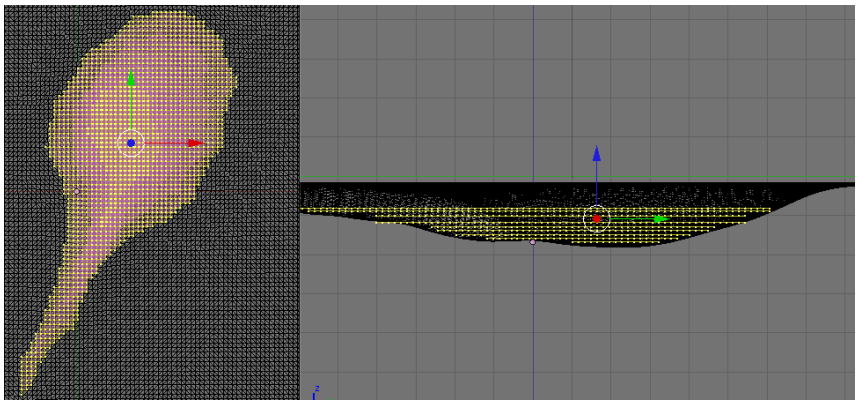


Obrázek 31: Skladba mostu a altánku

Vzhledem k reálným podmínkám jsou objekty statické. Materiál jsem se snažil přizpůsobit dřevu nastavil jsem nízkou hodnotu odrazivosti a střední hodnotu tření.

Mimo pevných těles jsou v příkladě obsaženy tři objekty typu *NxFluid* simulující tekutinu konkrétně vodu. Jedná se o:

- Jezírko – tekutina bez emitoru částic, to znamená, že odnikud nevytéká. Proto bylo nutné vytvořit síť bodů, jež budou reprezentovat dané částice kapaliny. Síť jsem modeloval podle terénu v několika vrstvách a importoval pomocí Wavefront formátu.

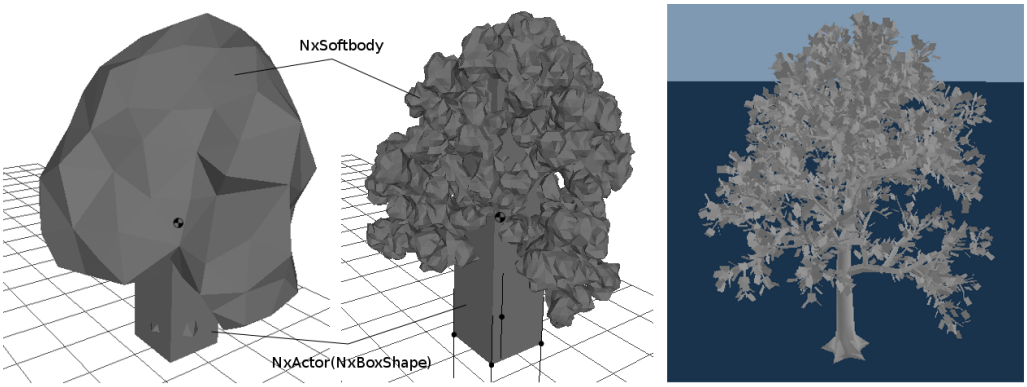


Obrázek 32: Vrstvení vodní masy

- Vodní proud potůčku - přítok je simulován pomocí emitoru (*NxFluidEmitter*), který jsem vytvořil nad objektem typu *NxFluid*.
- Déšť - je řešen stejným způsobem jako potok. Rozdíl je v umístění aktéra, ze kterého proud vytéká a směr toku.

Příklad zahrnuje i zástupce objektu tkanin jsou jimi jednoduché prapory umístěné na altánku a složitější tvar tkaniny reprezentující kimono či druh šatstva, je připevněno přímo na postavu sochy. Toto spojení reprezentuje můj pokus mapování složitějších tvarů tkanin na pevná tělesa (loutky), bez speciálních nástrojů.

V příkladu se vyskytují také stromy. Tento typ objektu jsem implementoval jako instanci třídy *NxSoftbody* tedy měkkého tělesa. Při implementaci jsem narazil na problém s kmenem, který byl příliš měkký a prohýbal se. Problém byl vyřešen připevněním pevného tělesa k měkkému tělesu v oblasti kmenu. Pro pozdější testování hardware jsem vytvořil modely stromů ve dvou stupních kvality rozdíl je patrný na obrázku 33.



Obrázek 33: Řešení stromů v PhysX

Obrázek 33 vlevo a uprostřed je vnitřní vrstva typu měkkého tělesa. Na každý vnější vrchol tohoto modelu se aplikují body trojúhelníkové sítě tak, že bude reagovat na změny polohy bodů z vnitřní vrstvy. Obrázek vlevo je konečná prezentace. Vzhledem k velmi omezenému popisu této problematiky v manuálu, jsem pro tento poslední krok použil již implementované třídy z PhysX tutoriálu.

Posledním prvkem je simulace větru. Pro tento účel je nejvhodnější třída silového pole (*NxForceField*). Popisné třídy měkkých těles a tkanin sice vlastní parametr *setExternalForce*, který má rovněž simulovat vítr, ale nevztahoval by se na další prvky scény. Vytvořil jsem tedy silové pole tvaru krychle, která obklopuje celou scénu, nastavil vektor působení konstantní síly na nízkou hodnotu a jelikož síla větru obvykle značně kolísá, přidal jsem vysokou složku náhodného působení síly ve všech osách.

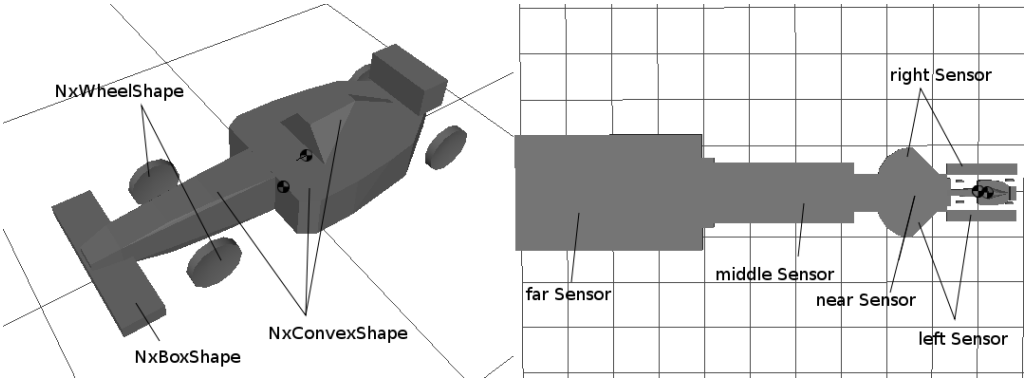
6.5 Příklad F1 Race

Posledním příkladem demonstruji použití speciálního tvaru - kola (*NxWheelShape*) a spínače událostí (*NxUserTriggerReport*). Jak již vyplývá z názvu, předmětem příkladu je závod formule 1. Vozy samostatně krouží po ohraničené trati a automaticky zatáčejí při blížící se zatáčce a vyhýbají se jiným vozům.

Obrázek 34: Příklad F1 Race

Jako terén trati jsem opět využil síť trojúhelníkových polygonů tj. instanci třídy *NxTriangleMesh* s materiály asfaltu a trávy. Tráva má shodné nastavení jako v předchozím příkladě a asfalt definuji s podobnými vlastnostmi jako skálu, rovněž z předchozí úlohy. Dalšími aktéry tvořící trať jsou vnitřní a vnější bariéry. Jsou vytvořeny jako samostatní aktéři, aby mohly senzory formule určit, kdy se blíží zatáčka a kterým směrem zatočit.

Formule je tvořena několika díly v rámci jednoho aktéra. Viz obrázek 35



Obrázek 35: Části formule

V pravé části obrázku 35 je zobrazen systém senzorů. Všechny mají nastavenou nulovou hmotnost, aby nijak neovlivňovaly vyváženost vozidla. Senzory near, middle a far slouží pro detekci vnitřní nebo vnější bariéry trati a podle toho volí směr zatáčení. Senzory left a right reagují jen na ostatní vozidla. Nalezení univerzálního a bezchybného řešení řízení tohoto typu by bylo obtížné. Model, který jsem navrhl není bezchybný, přesto naznačuje možné řešení problému.

- far Sensor (mírná zatáčka) – mírně sníží točivý moment vozidla a pootočí kola o $10°$,
- middle Sensor (ostřejší zatáčka) – sníží točivý moment vozidla o polovinu a pootočí kola o $30°$,
- near Sensor (velmi ostrá zatáčka – hrozí náraz) – točivý moment je snížen na minimum potřebné k jízdě a kola jsou pootočena o $45°$,
- left Sensor – pouze pootočí kola doprava o $30°$,
- right Sensor – pouze pootočí kola doleva o $30°$.

Dalším problémem při řešení jízdních vlastnostech formule byla přílišná vratkost vozidla ve větších zatáčkách. Při vyšší rychlosti stačila i menší zatáčka, aby se formule převrátila. Tento problém byl vyřešen snížením těžiště podle funkce z kapitoly 5.5.2.

7 Test a srovnání řešení za použití PhysX a CPU

Aplikace PhysX jsou primárně určeny pro novější řady grafických karet nVidia s podporou technologie CUDA, které dle dokumentace napomáhájí k plynulému chodu těchto aplikací. Pokud daný grafický adaptér není k dispozici, umožňuje engine PhysX přenechat veškeré výpočty na procesoru.

Cílem testovacích úloh je porovnat plynulost vytvořených PhysX aplikací běžících na několika výkonnostně odlišných procesorech bez grafického adaptéru podporující PhysX s plynulostí při běhu aplikace na specializované grafické kartě.

Jako testovací nástroj nabízí nVidia software AgPerfMon, který jsem použil při testování vytvořených úloh. AgPerfMon je program založený na záznamu událostí odehrávajících se v testované aplikaci. Tyto záznamy událostí dokáže zobrazovat v grafech a následně exportovat jako soubor *.csv* nebo jako obrázek grafu.

Testování jsem prováděl na těchto procesorech:

1. Intel Core2 Duo E4300, 1.8GHz, 2MB Cache, 800MHz FSB, s podporou grafické karty nVidia GeForce GT 440 (v grafech značeno jako GPU GT440),
2. Intel Core2 Duo E4300, 1.8GHz, 2MB Cache, 800MHz FSB, bez podpory grafické karty nVidia GeForce GT 440 (v grafech značeno jako CPU E4300),
3. Intel Pentium Dual-Core E5200, 2.5GHz, 2MB Cache, 800MHz FSB (v grafech značeno jako CPU E5200),
4. Intel Pentium Dual-Core Mobile T2080, 1.73 GHz, 1M Cache, 533 MHz FSB (v grafech značeno jako CPU T2080).

Již vím, že PhysX umožňuje vytvářet několik druhů objektů a které vlastnosti je charakterizují. V následujících úlohách jsem otestoval nároky na výkonnost jednotlivých typů objektů a porovnal výsledky na jednotlivých testovacích sestavách.

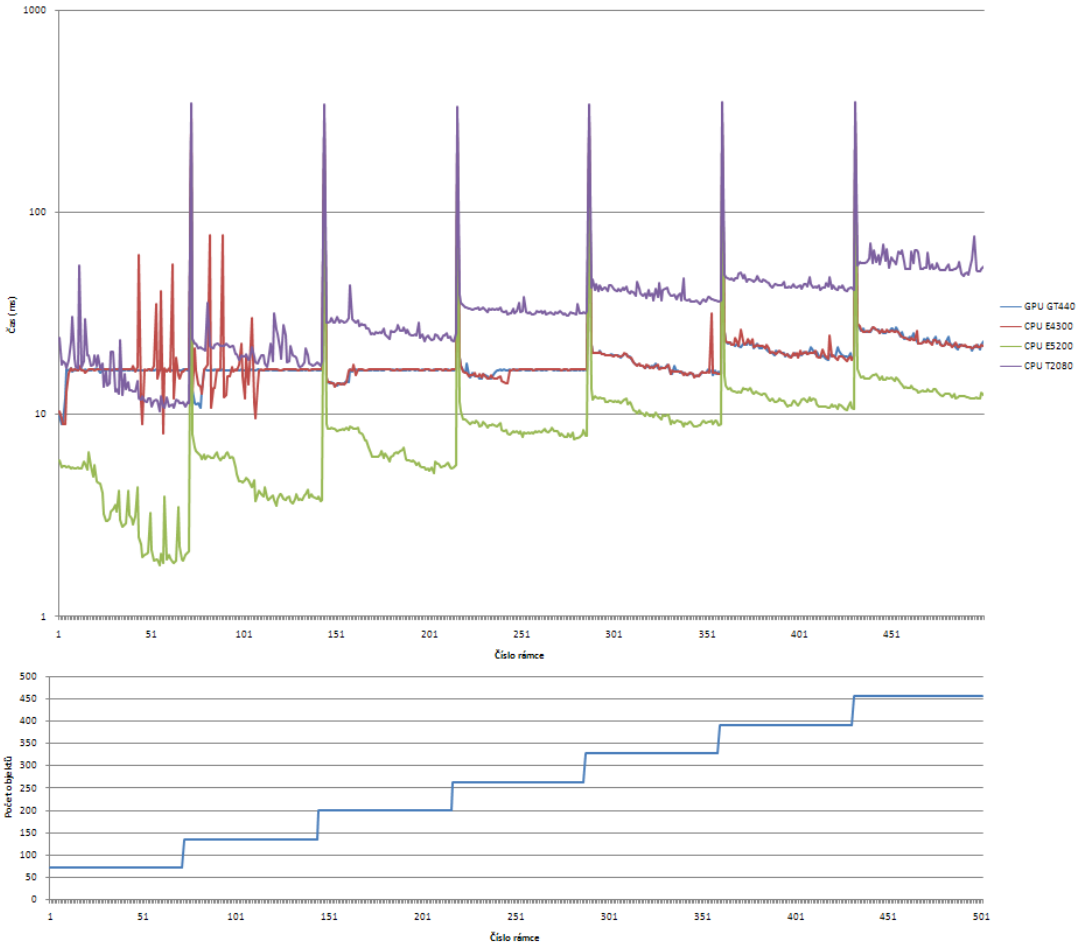
7.1 Test pevných těles

Test jsem prováděl na prvním vytvořeném příkladě. Na každé sestavě jsem spustil program se shodnými parametry a v konstantní časové periodě přidával na scénu další sadu aktérů. Vzhledem k tomu, že podpora grafické karty umožňuje akcelerovat pouze objekty typu *NxSoftBody*, *NxCloth* a *NxFluid* očekával jsem shodné výsledky na první a druhé testovací sestavě.

Výsledky testů prezentuje tabulka 2 a graf na obrázku 36. Nejlepšího výsledku dosáhla třetí sestava a první a druhá skončila dle očekávání téměř se stejnými výsledky. Na grafu je vidět náhlé nárůsty času, které jsou způsobené hromadným vytvářením nové sady aktérů.

	Průměrný čas snímku (ms)			
počet aktérů (n)	**1. sestava**	**2. sestava**	**3. sestava**	**4. sestava**
64	20,21	21,72	5,93	20,51
128	20,20	21,86	7,21	25,39
192	16,47	16,49	6,80	26,72
256	20,49	20,25	10,92	37,52
320	21,66	21,83	12,68	44,69
384	24,71	24,83	14,43	49,34
448	24,29	24,18	13,76	57,32
$\bar{n}$	21,15	21,60	10,25	37,36
	FPS			
$\bar{n}$	47,27	46,29	97,54	26,77

Tabulka 2: Test výkonu u pevných těles



Obrázek 36: Graf testu výkonu u pevných těles

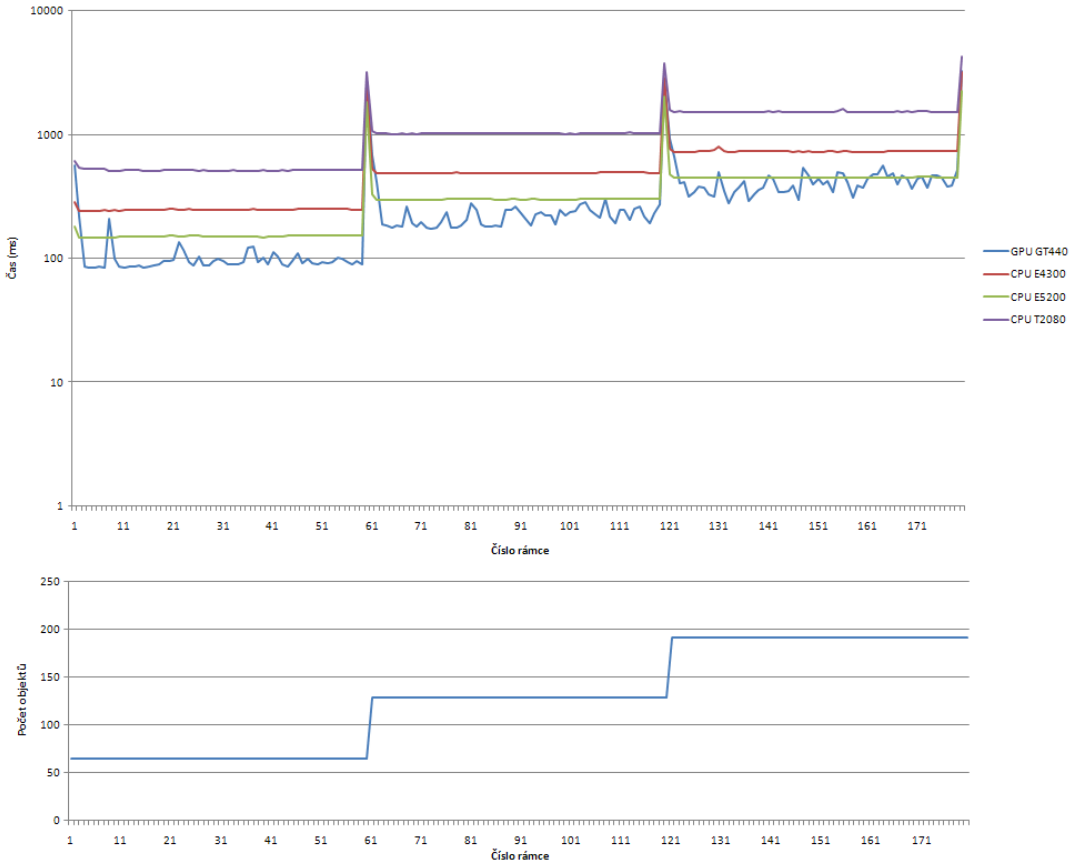
7.2 Test měkkých těles

Dalším testem bylo porovnání výkonu sestav při simulaci měkkých těles. Jako testovací příklad sloužil opět první program s pravidelným zvyšováním zátěže prostřednictvím nových objektů. V tomto případě jsem očekával významný rozdíl výkonu první a druhé sestavy, jelikož objekty typu *NxSoftBody* jsou akcelerovány pomocí GPU.

Výsledek opět dopadl dle očekávání, jelikož první sestava vybavená grafickým akcelerátorem podporující PhysX dosáhla významného rozdílu oproti sestavě druhé a mírného rozdílu proti sestavě třetí. Podrobnější výsledky znázorňuje tabulka 3 a graf na obrázku 37.

	Průměrný čas snímku (ms)			
počet aktérů (n)	**1. sestava**	**2. sestava**	**3. sestava**	**4. sestava**
64	151,69	243,31	152,01	471,87
128	276,49	539,82	335,85	1080,03
192	474,30	788,46	490,10	1593,33
$\bar{n}$	300,83	523,86	325,99	1048,41
	FPS			
$\bar{n}$	3,32	1,91	3,07	0,95

Tabulka 3: Test výkonu u měkkých těles



Obrázek 37: Graf testu výkonu u měkkých těles

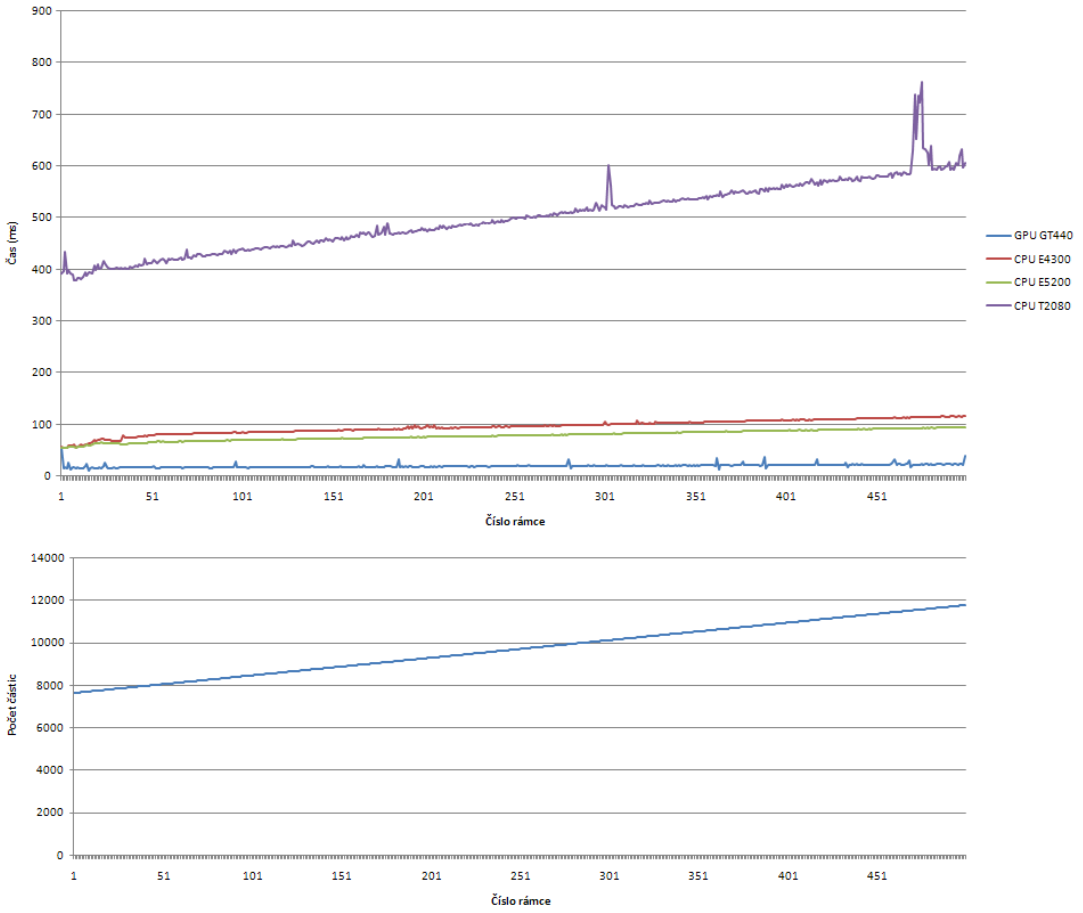
7.3 Test tekutin

Pro tento test jsem zvolil příklad Japanese Garden, kde simuluji pouze terén v nízké kvalitě, objekt jezera a potoku. Ostatní prvky byly vypnuty z důvodu neovlivňování testu.

V tomto testu suverénně zvítězila první sestava. Dosáhla více než čtyřnásobného výkonu oproti třetí sestavě, která skončila jako druhá v pořadí. Rozdíl výkonu první a druhé sestavy je v tomto testu téměř pětinásobný. Podrobnosti testu viz tabulka 4 a graf na obrázku 38.

	Průměrný čas snímku (ms)			
počet částic (n)	**1. sestava**	**2. sestava**	**3. sestava**	**4. sestava**
$\bar{n}$	19,59	95,05	77,87	498,83
	FPS			
$\bar{n}$	51,02	10,52	12,84	2

Tabulka 4: Test výkonu u tekutin



Obrázek 38: Graf testu výkonu u tekutin

Na druhém a třetím testu se prokázalo, že grafický akcelerátor podporující technologii PhysX se významnou měrou podílí na výkonu PhysX grafických aplikacích.

8 Závěr

Ve své práci jsem zužitkoval veškeré vědomosti, které jsem nabyl při jejím vytváření. Před tvorbou této práce jsem neměl valné zkušenosti s vytvářením 3D grafiky a aplikací s fyzikálním enginem, přesto mne toto téma velmi zaujalo.

Úkolem této práce bylo seznámit se s možnostmi, které jsou dnes k dispozici v oboru fyzikálních enginů. Ač se jedná o poměrně mladý obor, platí pro něj stejný trend jako téměř pro všechny obory informatiky a tím je rychlý vývoj. Fyzikálních enginů dnes existuje celá řada a uživatel má bohatou možnost volby.

Prozkoumat architekturu a vlastnosti PhysX byl můj další úkol. Jedná se o rozsáhlý, kvalitně implementovaný a poměrně dobře zdokumentovaný softwarový balík, což mi usnadnilo práci při implementaci ukázkových příkladů použití.

Nvidia neopomněla ani potřeby testovacího nástroje. Software je přehledný a snadno použitelný. Jedinou slabinou je dokumentace produktu. Samotné testy ukázaly převahu použití specializované grafické karty nad obecnými procesory ve všech oborech podporované akcelerace.

9 Reference

[1] *Physics engine - Wikipedia, the free encyclopedia* [online].
2005-01-31, poslední revize 2011-04-02 [cit. 2011-01-07].
<http://en.wikipedia.org/wiki/Physics_engine>

[2] *Ageia - Wikipedia, the free encyclopedia* [online].
2005-05-09, poslední revize 2011-02-12 [cit. 2011-01-12].
<http://en.wikipedia.org/wiki/Ageia>

[3] BURNES, Andrew. *AGEIA Acquires Meqon* [online]. 2005-09-01 [cit. 2011-01-12].
<http://ve3d.ign.com/articles/news/8437/AGEIA-Acquires-Meqon>

[4] *Ageia PhysX PPU - PhysX Wiki* [online].
2011-01-16, poslední revize 2011-04-26 [cit. 2011-02-05].
<http://physxinfo.com/wiki/Ageia_PhysX_PPU>

[5] *PhysX - Wikipedia, the free encyclopedia* [online].
2005-05-09, poslední revize 2011-04-19 [cit. 2011-01-13].
<http://en.wikipedia.org/wiki/PhysX#Nvidia_acquisition>

[6] *Physics processing unit - Wikipedia, the free encyclopedia* [online].
2005-05-09, poslední revize 2011-01-09 [cit. 2011-01-07].
<http://en.wikipedia.org/wiki/Physics_processing_unit>

[7] BLACHFORD, Nicholas. *Lets Get Physical: Inside The PhysX Physics Processor* [online].
c2006 [cit. 2011-01-07]. <http://www.blachford.info/
computer/articles/PhysX2.html>

[8] NVIDIA Corporation, *PhysX Documentation*, Santa Clara, c2008

[9] *Datový typ - Wikipedie* [online].
2006-02-10, poslední revize 2011-01-18 [cit. 2011-02-19].
<http://en.wikipedia.org/wiki/Physics_processing_unit>

[10] *Abstraktní datový typ - Wikipedie* [online].
2005-08-19, poslední revize 2011-01-31 [cit. 2011-02-19].
<http://cs.wikipedia.org/wiki/Abstraktní_datový_typ>

[11] *Setting Up Your PhysX Environment* [online].
2010-08-24 [cit. 2010-12-18]. <http://gputoaster.wordpress.com/
2010/08/24/setting-up-your-physx-environment>

[12] ŠTEFEK, Petr. *PhysX aneb akcelerovaná fyzika ve hrách* [online].
2008-08-20 [cit. 2011-02-20]. <http://www.svethardware.cz/
art_doc-AFFA91FDDC04437AC12574A4006DB634.html>

[13] REICHL, Jaroslav; VŠETIČKA, Martin. *MECHANIKA TUHÉHO TĚLESA :: MEF :: Encyklopedie Fyziky* [online]. c2006–2011 [cit. 2011-03-07]. <http://fyzika.jreichl.com/index.php?sekce=browse&page=82>

[14] *OpenGL - Wikipedie* [online]. 2005-01-09, poslední revize 2010-08-11 [cit. 2011-01-07]. <http://cs.wikipedia.org/wiki/OpenGL>

[15] TIŠNOVSKÝ, Pavel. *Grafická knihovna OpenGL (1)* [online]. 2003-07-01 [cit. 2011-01-28]. <http://m.root.cz/clanky/graficka-knihovna-opengl-1/>

[16] *Wavefront OBJ File Format Summary* [online]. [cit. 2011-03-16]. <http://www.fileformat.info/format/wavefrontobj/egff.htm>